Why your back hurts
-
Finding an efficient way to measure and evaluate sitting posture using a combination of body sensors placed on the body and machine learning

Gijs van Rhijn
S1721690
Bachelor assignment
Creative Technology
University of Twente
July 2019

Supervisors: Angelika Mader, Gwenn Englebienne

Abstract

The goal of this paper is to test the effectiveness of various wearable sensors, wearable sensor positions, and machine learning for measuring posture quality. Various different seating postures were identified. These where divided into correct and incorrect postures. Measurements of people taking on these different postures where done using a sensor suit containing various stretch sensors, gyroscopes, and accelerometers in different positions. The data gathered during these measuring sessions was used to train the random forest machine learning algorithm. The variable importance measures of the random forest algorithm where used to identify the most important sensors and sensor positions. Using about 300 lines of data, the random forest algorithm can be trained to have an average accuracy of about 92 per cent. Adding random rotations to the sitting position (to simulate a real office setting) decreases said accuracy to 84 per cent. The variable importance estimations placed high importance on the shoulder and head placed gyroscopes. Using the data from only these sensors results in only a slight reduction in accuracy to 81 per cent.

Acknowledgements

I would like to thank my two supervisors, Angelika Mader and Gwenn Englebienne. Their willingness to have weekly progress meetings, despite their busy schedule, helped me stay focused the past few months. Their advice helped me solve the many problems I encountered while working on my thesis

I would also like to thank my friends and acquaintances for helping me out during the various measuring sessions. They were always willing to take some time out of their day to help me gather the necessary data to complete my thesis.

Table of Contents

Abstract ... **3**

Acknowledgements ... **5**

Table of Contents ... **7**

1 Introduction ... **10**

1.1 Context ... 10

1.2 Goal of report ... 10

1.3 Research questions ... 10

1.4 Setup of report ... 11

2 State of the art ... **12**

2.1 Correct and incorrect posture ... 12

2.1.1 Interview with physio therapist ... 12

2.1.2 Interview with orthopaedic surgeon ... 13

2.1.3 Further studies ... 15

2.1.4 Summary and overview incorrect postures ... 15

2.2 Machine Learning ... 16

2.2.1 Literature research: Machine learning techniques suitable for posture identification ... 17

2.2.2 Literature research: Variable importance ... 18

2.2.3 Literature research: Conclusion ... 18

2.2.4 Random Forests ... 19

2.3 Similar products ... 21

2.4 Conclusion ... 21

3 Ideation ... **22**

3.1 Sensors ... 22

3.1.1 Sensors for each type of bad posture ... 22

3.1.2 xSens ... 25

3.1.3 Types of sensors that will be used ... 26

4 Specification ... **28**

4.1 Sensors ... 28

4.1.1 Sensor modules that will be used ... 28

4.2 Suit ... 28

4.2.1 First version sensor suit (used in session one and two) ... 28

4.2.2 Second version sensor suit (used in session three) ... 29

4.2.3 Third version sensor suit (used in session four) ... 29

4.3 Machine learning ... 30

4.3.1 Data processing ... 30

4.3.2 Machine learning software 30
4.4 measurement design 30
4.5 Consent form and ethical approval 30
5 Realisation. 31
5.1 Pilot measuring session. 31
5.1.1 Result from the pilot measuring session. 31
5.1.2 Improvements made based on the pilot measuring session. 31
5.2 First measuring session. 31
5.2.1 Improvements based on the first measuring session. 32
5.3 Second measuring session. 32
5.3.1 Improvements based on the second measuring session. 32
5.4 Third measuring session. 32
5.4.1 results from the third measuring session. 32
5.4.2 improvements based on the third measuring session. 33
5.5 Fourth measuring session. 33
5.5.1 results from the second measuring session. 34
6 Data Analysis 35
6.1 Data analysis session one 35
6.1.1 Training the random forest model using unprocessed data 35
6.1.2 Training the random forest model using data relative to the correct posture 36
6.1.3 Analysing variable importance. 37
6.2 Data Analysis session two 38
6.2.1 Training the random forest model using data relative to the correct posture. 38
6.2.2 Analysing variable importance 39
6.3 Data Analysis session three 40
6.3.1 Training the random forest model using unprocessed data 40
6.3.2 Training the random forest model using data relative to correct posture 40
6.3.3 Training the random forest model using data relative to the upper back sensor. 41
6.3.4 Analysing variable importance 42
6.3.5 Simplifying the machine learning model 43
6.4 Data analysis session four. 45
6.4.1 Training the random forest model using unprocessed data. 45
6.4.2 Training the random forest model using data relative to the correct posture. 45
6.4.3 Analysing variable importance. 46
6.4.3 Simplifying the machine learning model 46
7 Discussion. 50

7.1 Discussing the results. 50
7.1.1 Effects of processing the data. 50
7.1.2 Simplifying the sensor suit. 50
7.2 Recommendation for further research. 51
7.3 Conclusion. 52
8 References 54
9 Appendix 57
Appendix A, python machine learning code. 57
Appendix B, Arduino code sensor suit. 59
Appendix B-I, first version of the Arduino code. 59
Appendix B-II, second version of the Arduino code. 67
Appendix B-III, third version of the Arduino code. 74
Appendix C, checklist ethical approval 82
Appendix D, consent form 86
Appendix E, information sheet 88
Appendix F, picture of the real sensor suit 92
Appendix G, reflection report on ethical impact bachelor project 93

1 Introduction

1.1 Context

Many people in the western world are living a sedentary lifestyle [1]–[6]. According to Spittaels et al. [3], people in Belgium spend 55% of their waking time in sedentary behaviour. Spending significant time sitting is associated with back problems [7]. A person's posture while sitting can influence the back problems they have a tendency to experience. Specifically lower back pain is common [8]. Improving one's posture can have significant effect on the back problems they experience, both now and in the future.

Nowadays, people seem to becoming more and more aware of their health and the consequences the modern sedentary lifestyle has on their bodies. Health focused smart wearables are becoming more popular each day. These wearables give people access to their activity levels, sleep schedules, heartrate, and more. Posture is a common cause of concern amongst people. Many people have trouble keeping an upright posture during the day. This creates problems like low back pain and neck pain. Posture is also something that can be important for people with specific skeletal deformities or particularly weak backs. For these people keeping track of their posture can be even more important to their wellbeing. To help people keep track of their posture, a wearable measuring device would be welcome. Such a device could give people real time feedback on their posture, but it could also be used by medical professionals to do research about the effect of faulty postures.

Precise posture measurement tends to be a difficult thing to do due to the myriad of ways in which a posture can be incorrect. Machine learning (ML) is a promising tool to solve problems that are as complicated as posture measurement. ML techniques can find patterns in complicated, highly dimensional, data. This might give machine learning an advantage over more traditional programming techniques. Machine learning has one big strength in detecting variable importance. Historically, determining the importance of a specific variable has been quite complicated. The ability of ML techniques to create order in a complicated data set, allows for new possibilities in determining variable importance. Often these techniques are seemingly more accurate that more traditional techniques. Detecting variable importance can be helpful when trying to minimize the complexity of a posture sensing device, hopefully allowing the cost and complexity of the device to be reduced.

1.2 Goal of report

This GP is about the effectiveness of using machine learning and different wearable sensors to measure sitting posture. The idea is that this could help people keep track of, and improve, their posture. The goal of this paper is to identify suitable machine learning techniques to help identify sitting posture and to help identify what sensors and sensor placement are important.

One or more promising machine learning techniques will be identified, together with various types of wearable sensors the effectiveness at posture measurement will be tested. The importance of individual sensors and sensor placement can be tested using the build in variable important measures of the machine learning techniques. Then the effectiveness of the machine learning techniques without these sensors can be tested in an attempt de declutter the system.

1.3 Research questions

The main research question of this report is as follows:

- *How to efficiently measure and evaluate sitting posture using a combination of sensors placed on the body and machine learning?*

The sub questions are:

- *What constitutes good and bad posture?*
- *What types of sensors can be used to measure posture problems?*
- *What type of machine learning is best used?*

- *How many sensors are needed, and where on the body do they need to be placed to get reliable and accurate posture measurements?*
- *How effective is the chosen machine learning algorithm at achieving the goal of measuring posture?*

1.4 Setup of report

This report will start with an overview of the theoretical background necessary to design and execute a proper test into posture measurement. This information can be found in chapter 2, “state of the art”. This chapter will explore what makes a sitting posture good or bad, and identify different kinds of bad and good posture. It will also create an overview of different machine learning techniques that might be useful for posture identification. Lastly, it will take a look at commercial posture sensors that are available already and it will explore the advantages and disadvantages of these devices.

in chapter 3, “ideation”, different types of sensors will be discussed and a final list of useful sensors will be constructed. This list is then translated to real life sensor modules in chapter 4, “specification”. The specification chapter also describes the design of the sensor suit and what machine learning software will be used. Lastly, chapter 4 will detail the way the actual tests were executed.

Chapter 5, “realisation”, serves to give the reader a quick overview of the different measuring sessions that have been done and their results. For a more detailed analysis of the data gathered during each testing cycle, see chapter 6, “data analysis”. Chapter 7, “discussion”, acts as a concluding chapter. In this chapter the most important results from the various tests will be discussed, recommendations for further research will be given, and there will be a reflection on the project as a whole. Finally, the report will finish with a chapter for references and one for the included appendices.

2 State of the art

2.1 Correct and incorrect posture

Before a posture measuring device can be designed and tested, it is important to know what defines a posture to be good or bad. The exact consequences of different seating postures do not seem to be understood perfectly. Some researches view the connection between posture and back problems as a largely empirical one [9]. Still, research regarding the biomechanics of the human body date back hundreds of years [10]. In an attempt to create an overview of the knowledge available, Empirical and biomechanical knowledge will be combined with data from cross-sectional studies.

Many healthcare professionals have years of experience with treating back and neck problems. This experience can serve as a basis for a proper understanding of postures. According to an interview with a physiotherapist described by Visser [11], posture quality is largely dependent on the position of the pelvis (Figure 1).

2.1.1 Interview with physio therapist

To obtain a correct posture, the pelvis needs to be rotated forwards. This allows for the lumbar to develop a Lordosis. The lumbar lordosis results in a kyphosis in the thoracic part of the spine. Finally, the cervical part of the spine will have a lordosis again, this allows for the head to be positioned correctly on top of the spine. This allows the head to balance nicely on top of the spine and for the scapulae to be positioned straight on the back. This shape of the spine allows it to act like a spring, giving it the ability to correctly handle the pressures placed on it.

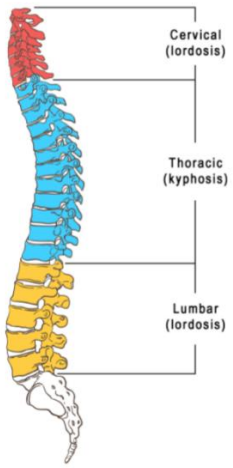


Figure 1, Cervical, Thoracic and Lumbar part of the spine [11]

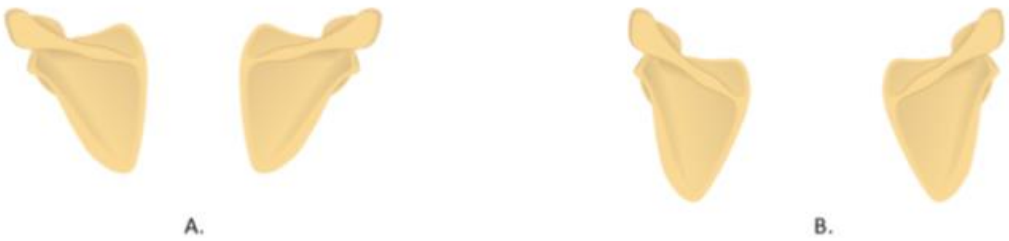


Figure 2, correct scapulae posture (A), incorrect scapulae posture (B) [11]

Using the information about a correct posture as a basis, it is possible to explain what makes an incorrect posture. Similar to the correct posture, an incorrect posture starts with the pelvis. When the pelvis is tilted backwards instead of forwards, the lumbar lordosis disappears. Instead the lumbar is pulled backwards. To compensate for the shape of the lumbar, the thoracic part of the spine moves

forwards. This shape of the spine ultimately pushes the head forwards, forcing the chin to rotate upwards to allow the head to look straight.

2.1.2 Interview with orthopaedic surgeon

To expand upon the knowledge acquired by Visser [11], an interview with an orthopaedic surgeon was done. This interview focused on expanding on the information described by Visser. It gives a greater insight into de details of correct and incorrect posture.

A correct posture focusses on efficient use of the muscles required to maintain said posture while preventing unnecessary stress on the vertebrae and joints. A posture can be described from three different perspectives: the coronal plane, the sagittal plane, and the transverse plane (Figure 3**Error! Reference source not found.**). Visser gave an overview of the correct and incorrect posture in the sagittal plane. The straightening of the spine allows the back muscles to efficiently keep the body upright. The natural curve of the spine helps balance the centre of gravity above the pelvis. A slouched forward posture, with the pelvis tilted backwards, puts stress on the vertebrae and joints to keep the body balanced instead of using the muscles. This effect is also described by Corlett and Eklund in their paper *how does a backrest work* [12]. When sitting straight int the sagittal plane it is important to keep one's head straight above their shoulders and pelvis, as to not put any unnecessary pressure on it.

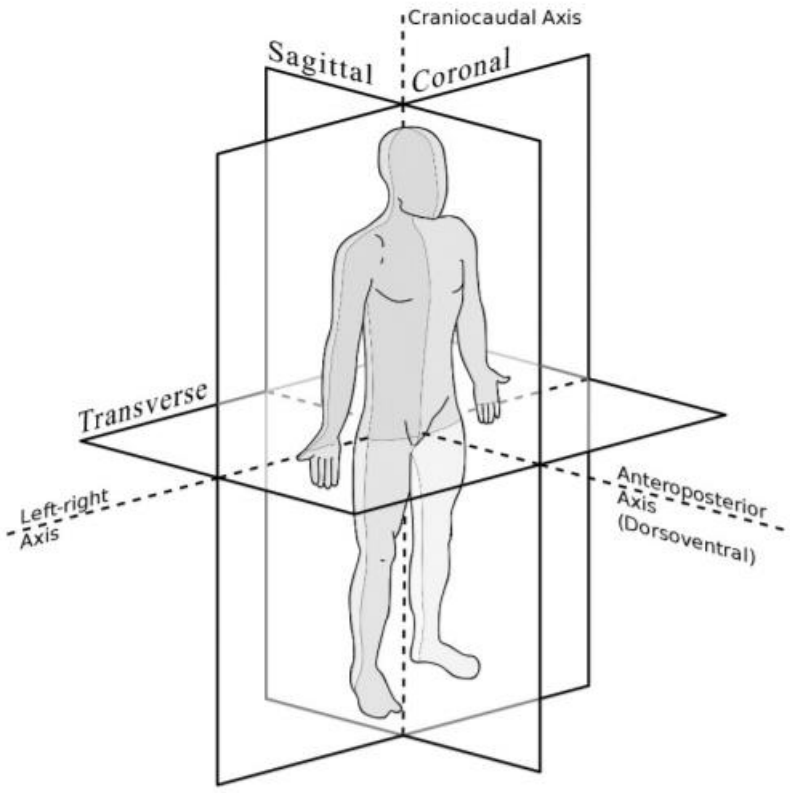


Figure 3, the coronal, sagittal and transverse plane. [13]

A similar reasoning can be used to evaluate the coronal and transverse plane. To maintain a healthy posture the shoulders should be at equal height in the coronal plane. This keeps the body balanced. The spine and pelvis should also be straight in this plane, thus not be slanted to one side. (see Figure 5) In the transverse plane it is important to keep the shoulders above the hips, instead of them being

rotated. This rotation is also important for the spine. The spine vertebrae should not be angles in relation to each other.

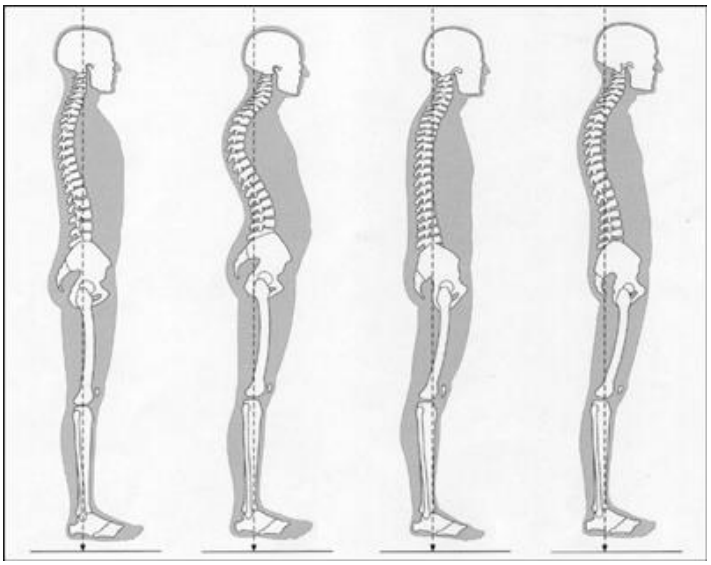


Figure 4, postures in sagittal plane, the leftmost posture is correct. [14]

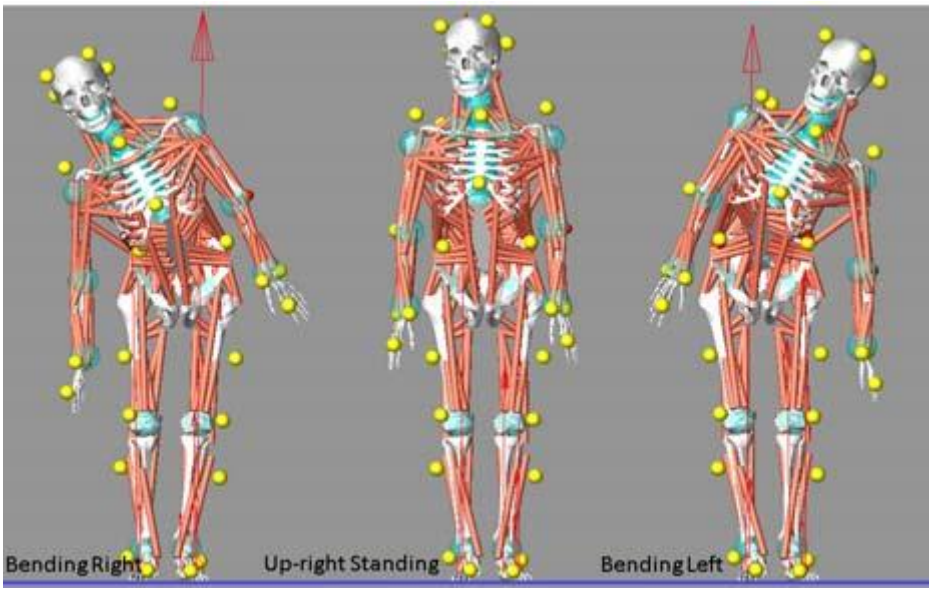


Figure 5, postures int he coronal plane. [15]

Unnecessary pressure in the body due to improper posture can result in pain. A completely static posture tends to have a negative effect on the body as well as it can put continuous strain on certain parts of the body. One technique that is meant to result in a more dynamic posture is to sit on a chair or stool that is not completely stable. This forces the body to balance itself, creating a more dynamic posture.

Most complaints concerning pain in the trunk are about the lower back and the neck. The upper back tends to be less problematic. This can be explained by looking at the effect of the ribs on the stability of the upper back. The ribs give the upper back a certain amount of stability in return for a decrease of the freedom of motion. Because of this it is easier for to keep a stable upper back without causing unnecessary strain on the body.

The muscles responsible for keeping the correct posture can be divided into the extrinsic (large) muscles and intrinsic (small) muscles. The extrinsic muscles are responsible for the general shape of the body. The muscles in the back of the body determine position of the pelvis and the shape of the spine. The abdominal muscles can tense to form a 'hard balloon' like structure. This helps balance the back. The intrinsic muscles are responsible for the position of the vertebrae relative to each other. They prevent them from shifting or rotating away from each other. Extrinsic muscles are relatively easy to train and measure, while the intrinsic muscles are so small that it becomes very hard to accurately measure and train them.

As mentioned before, the goal of a correct posture is to minimize the amount of problems people experience in their back and neck. Human are unique amongst vertebrates in that they walk on two legs. To facilitate standing and walking upright, humans have the 'hollow' lordosis curve in the lumbar part of their spine. Still, the general structure of the spine is not very well suited to an upright posture. Because of this, back and neck pains in humans are common. Nearly everyone will experience them in their life. Back problems can be acute, but they can also be chronic. Chronis back problems tend to be extremely hard to cure. It is possible for chronic back problems to exist without any direct biological cause, this is called chronic pain syndrome.

Not everybody is the same, this is also true for one's skeleton. Every person will have a slightly different 'correct' posture. Some people have deformed spines. This can be the results of different causes. The deformity can be congenital, but it can also be caused by a growth disturbance later in life. Spinal deformities can also be the result of a traumatic experience. When such a person's posture is measured the standard rules for good and bad posture cannot be applied. It might be impossible for these people to attain correct posture. Because of this, their posture measurements have to be adapted to their body. The correct posture for people with a non-normal back has to be determined by a medical professional, as this can be quite an intricate task.

2.1.3 Further studies

Various studies have found posture to be a moderate to strong predictor of low-back pain. Adams, Mannion, and Dolan [16] found the amount of lumbar lordosis to be a consistent predictor of low back pain. Christie, Kumar, and Warren found the shape of the spine to be a moderate predictor of low back pain. They state the importance of other, unidentified, predictors of low back pain. The findings of both studies match what was said in the interview done by Visser with the physiotherapist and the interview that expands on this with the orthopaedic surgeon.

Other than low back pain, neck pain is also considered to be a common risk of bad posture. Brink et al. [17] found that increased head flection predicted a higher pain score on upper quadrant musculoskeletal pain. The upper quadrant is a part of the body that includes the neck. Studies by shows that increased flection of the neck increases the gravitational moment on the neck [18]–[20] . This increased neck flection can be a source of potential neck pain.

2.1.4 Summary and overview incorrect postures

A correct posture can be defined as any posture that allows for efficient use of the muscles required to maintain said posture while preventing unnecessary stress on the vertebrae and joints. Incorrect posture can be defined as any posture that is not a correct posture. Posture can be described from the perspective of three planes: coronal, sagittal and transverse. In the coronal plane it is important to keep your shoulders and spine straight. In the sagittal plane, the right position of the pelvis supports the lordosis in the lumbar part of the spine. This creates a slight curve in the spine, allowing it to adjust to the forces placed on it. Lastly, in the transverse plane, the vertebrae should be straight on top of each other. This includes them not being twisted in relation to each other.

To aid the development of a posture sensing device, a list of different faulty and correct postures was constructed. These postures can then be used to determine useful sensors for the measuring device. The postures can be seen below in Figure 6 till Figure 8

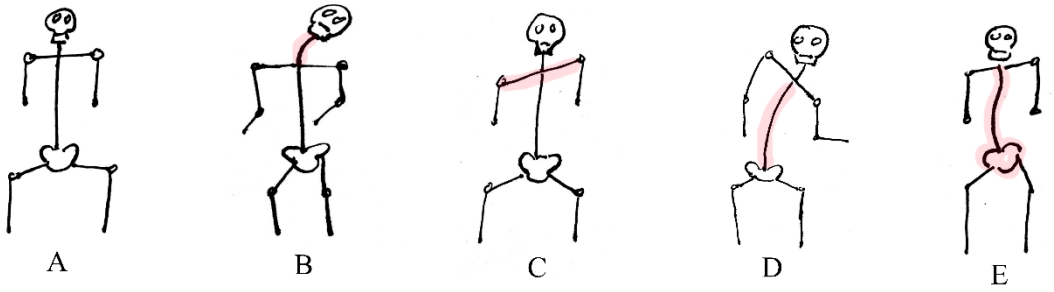


Figure 6, postures in the coronal plane. A: correct posture. B: neck bend sideward. C: shoulders not straight. D: correct pelvis with bend back. E: pelvis rotated sideways causing scoliosis in the back.

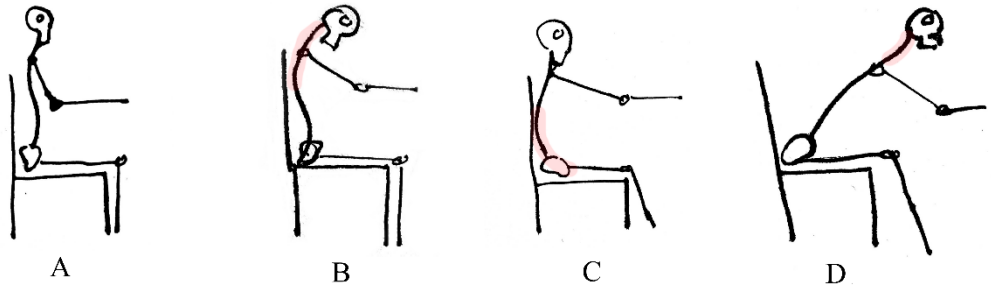


Figure 7, postures in the sagittal plane. A: correct posture. B neck and part of upper back bend downwards. C: pelvis rotated backwards, slouched posture. D: bend forwards, causing neck to be angled upwards.

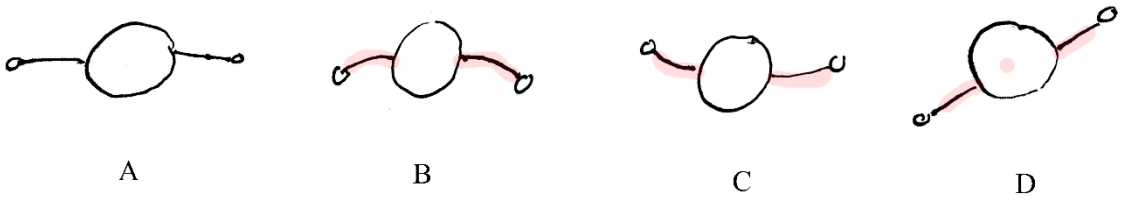


Figure 8, postures from the transverse plane. A: correct posture, shoulders straight above hips. B shoulders bend forwards. C: shoulders bend backwards. D: shoulders rotated, potentially causing the spine to rotate as well.

2.2 Machine Learning

Before the effectiveness of machine learning at posture measurement can be tested, one or more suitable machine learning techniques need to be identified. Identification of sitting posture is known as a supervised classification problem. Supervised machine learning problems focus on finding connections between different classes of training data. The training data has been pre-divided into classed before the model is trained. Classification problems focus on categorizing the input data in discrete classes, for example: good posture versus bad posture.

Many machine learning algorithms offer methods to estimate the importance of the various input variables. These techniques can be useful when attempting to identify the important inputs of a system. In the case of posture measurement, it can be used to identify the important sensors in a sensor

array. This allows for the unimportant sensors to be removed. Removing the complexity and the cost of the system.

This paper will first identify relevant supervised classification machine learning algorithms by looking at literature about posture measurement. These machine learning algorithms should be relatively simple to implement via third party software to make integration with the graduation project as smooth as possible. Then it will try to compare the effectiveness of these algorithms for the intended goal of posture measurement. Lastly it will compare the accuracy of variable importance measurements. The goal of this literature review is not to create and evaluate an exhaustive list of suitable machine learning algorithms. Rather, the goal is to identify one or more suitable algorithms that can be used during the real live tests. Because of the number of algorithms available, it is beyond the scope of this paper to make the list exhaustive.

2.2.1 Literature research: Machine learning techniques suitable for posture identification

Machine learning techniques can be divided into classification and regression problems. Supervised machine learning problems focus on finding connections between different classes of training data. The training data has been pre-divided into classed before the model is trained. Unsupervised machine learning algorithms on the other hand focus on finding new classes within data by looking at the underlying structures. Unsupervised algorithms can potentially identify new useful classes. Classification problems focus on categorizing the input data in discrete classes, for example: good posture versus bad posture. On the other hand, regression algorithms have a continuous output. An example of this would be estimating the price of a house in a specific neighbourhood. Identification of sitting posture is a supervised classification problem.

Many machine learning techniques exist that can be used for supervised classification problems. To identify promising techniques for posture measurement, a literature review is done concerning research about applying machine learning for similar goals. Various kinds of machine learning where used successfully in combination with sensors worn on the body. One study by Attal et al. [21] compared different machine learning techniques with wearable sensors. They attempted to use these techniques to try to human activities. Four different supervised classification machine learning techniques where used: K-nearest neighbor (K-NN), support vector machines (SVM), gaussian mixture models (GMM), and random forests (RF). Their research found the K-NN model to be the most accurate of the bunch. The SVM and RF models scored very close to K-NN and very similar to each other. The GMM scored significantly lower than the other 3, not even managing to break the 90 percent accuracy in any of the tests. Comparatively, the other 3 methods scored accuracy scores of close to 95 percent or more.

The findings by Attal et al. are confirmed by other researchers. Another research by Pansiot et al.[22] tested the effectiveness of GMM in activity tracking using an accelerometer attached to the ear. Even though the accuracy of the individual activities could get up to 100 percent, the average accuracy of these predictions turned out to be 83 percent. This number falls in line with numbers from the study done by Attal et al. A study by Patel et al.[23] that looks at activity tracking using random forests with wearable sensors shows an average accuracy of around 90 percent. This is slightly lower than the study by Attal et al. discussed earlier. It is, however, still higher than all the results discussed about GMM. In a study by Rodriguez-Martin et al.[24] SVM is shown to have an accuracy of more than 90 percent when identifying activity type, using only a single accelerometer attached to the waist. Another study into human activity recognition by Altun and Barshan shows an accuracy of over 90 percent for both SVM and K-NN. SVM and K-NN again show great similarity to each other. The findings of these studies match the findings of Attal et al. Based on the studies shown,

there is strong evidence that SVM, K-NN and RF are capable candidates for interpreting data from body worn sensors. GMM is shown to be effective, but lacking compared to the other methods. The studies mentioned do not look at sitting posture measurement, instead they try to identify different types of human activities. For this reason, the results do not apply directly to the goal of this paper. However, the type of sensors used and the nature of the measurements is similar enough that the results still seem relevant. Thus, making K-NN, SVM, and RF promising tools for posture measurement.

2.2.2 Literature research: Variable importance

In the previous segment of this paper 4 different machine learning techniques where discussed. Of these four techniques, GMM performed significantly worse than the rest in tests using sensor setups similar to the ones that will be used in this GP. The other three techniques performed very similarly to each other, with accuracies of over 90%. Of these three techniques only one offers variable importance methods, namely the Random Forest[25], [26]. Random forests variable importance uses four different measures to determine the importance of the variables.

Variable importance is known to be a complicated problem. However, it can be of great help when trying to reduce the complexity of a model. In the case of posture measurement this would ideally reduce the number of sensors necessary. In turn, this would increase the wearability of the device, while decreasing the costs, complexity and power use. Experience with variable importance measures in random forests by Liaw and Wiener[27] has shown that it can be useful for reducing the complexity of a model. Similarly, research by Belgiu and Drăgut[28] showed the usefulness of RF variable importance when trying to reduce the complexity of a model. Still, the measures in random forests give imperfect indications of the variable importance. Research by Strobl et al.[29] using simulated data shows potential problems with the variable importance methods of random forests. According to Strobl et al. "…suboptimal predictor variables may be artificially preferred in variable selection. "(P.1, Abstract)." The problems they found are mostly present in data with datasets that use data with a varying level of categories, for example using categorical and continuous data in the same random forest. The methods are shown to work well with data with a consistent level of categories. To solve the problem of variable importance, Strobl et al. present the cForest algorithm. In the simulations it is shown that the cForest algorithm attains significantly better results at the expense of computation speed. The cForest algorithm is available R[30], a popular programming language designed for statistical purposes.

All ML methods discussed above can be implemented relatively easily using 3rd party software libraries. Scikit learn[31] is a python Machine Learning library allowing for the implementation of all these algorithms. All algorithms being available in the same environments allows for easy switching between different types of machine learning, making it possible compare the different algorithms. This also makes it possible to easily use different types of machine learning for their individual strengths. For example, it is possible to use the RF algorithm to estimate variable importance and to then use this new, less complicated system with the other types of machine learning to try and get the best results.

2.2.3 Literature research: Conclusion

Four types of machine learning where discussed in this paper: K-NN, SVM, RF, and GMM. Of these four types, GMM was found to be the least effective when doing measurements using body mounted sensors. K-NN, SVM and RF where found to be similarly accurate, with accuracies greater than 90 percent. Random Forests is the only machine learning technique that is able to estimate variable importance, potentially helping to decrease the complexity of the measuring system. Because all machine learning algorithms are available in Scikit learn, easy switching between them is possible. This allows the RF variable importance measures to be used for simplifying the system, while still using other ML methods to processed the data

from this simplified system.

The types of measurements done in the literature examined int this paper focused on determining activity type, like sitting, walking, and running. This is a different type of measurement than posture measurement. Because of this the results of the found studies do not relate directly to posture measurement. However, the sensors used for the measurements where often gyroscopes and accelerometers. These are the main types sensors that are of interest when measuring posture. The sensors were also worn on the body; this again matches with the sensors that will be used for the posture measurements. Because of the matching measuring equipment, the research can be seen as an indication of the effectiveness of the various machine learning techniques for posture measurement.

The machine learning techniques discussed in this paper are not an exhaustive list of all the different techniques available, but it does provide a basis for further research into posture measurements. Further research could focus on identifying more machine learning techniques and comparing their effectiveness. Further research could also improve on the accuracy of the effectiveness estimations, giving a better idea of what machine learning techniques are most useful for posture measurement.

2.2.4 Random Forests

One popular supervised machine learning method that is suitable for classification is Random Forests [32]. Random forests are a machine learning technique developed by Breiman. Random forests are not only known to be good performers in a lot of situations, but also relatively user friendly.

The previous literature research showed great potential for the random forest machine learning algorithm. Radom forests is known to be a robust algorithm able to handle high dimensionality[28]. The ability of random forests to estimate variable importance can be extremely helpful when trying to decrease the complexity of the sensing system. Because of the potential of random forests for posture measuring, a more detailed overview of the technique was written

2.2.4.1 CART decision trees

Random forests are based on CART decision trees [32]. CART stands for classification and regression trees. As the name suggests, CART uses decision trees to perform classification or regression analysis. These decision trees are generated based on a set of labelled test data. Classification and regression trees use a different method to determine the right split for each node. Each node of a classification decision tree is split based on the GINI index. The GINI index is considered to be a greedy approach to splitting. This means that it tries to minimize the cost of running the tree by choosing what seems like the most efficient splits. The GINI index works by predicting the “purity” of a node. It does this by comparing the labels of the training data of each node [33]. If there is a large mix of different labels, the node is considered to be impure. A node of higher purity is considered to be more desirable.

Decision trees are known to be prone to overfitting. Overfitting is a problem where the decision tree fits the training data too well. This causes any unwanted noise present in the training data to be incorporated in the decision model. This decreases the accuracy of the model.

2.2.4.2 Bagging

In an attempt to increase the accuracy of CART machine learning and to decrease the chance of overfitting, Breiman developed the concept of Bagging [34]. Bagging is a type of assemble learning method. Assemble learning methods use multiple predictors, in the case of bagging these are decision trees, and aggregates the output [27]. Bagging works by creating multiple trees out of the training data, using only part of the training data to create each tree. The outcome of these multiple trees is then aggregated to decide the final output of the algorithm. This outcome is decided in a different way for classification and regression trees. When bagging is used for classification, the outcome is determined

by voting. Each tree in the algorithm "votes" for what it determined to be the right class. The class with the largest number of votes will be the final output of the algorithm. Regression algorithms work by outputting an average of the results of all the trees.

Bagging is closely related to random forests. Liaw and Wiener described random forests as "adding another layer to bagging" [27]. In random forest extra randomness is introduced by choosing random variables for each decision note in each tree made, instead of using all variables for each note. This randomness greatly helps to prevent overfitting [32] and allows for accurate predictions to be made.

2.2.4.3 Out of bag estimation

In his 1996 paper, Breiman introduces what he calls the 'out of bag estimation' (OOB) [35]. This is a technique that can be used to calculate an internal error rate when bagging decision trees. The bootstrap sample used for each tree uses about 67 percent of the training data, while leaving the other 33 percent unused. This other 33 percent can then be used to estimate statistics about the data. For example, an internal error rate can be calculated using an OOB estimation. In the real life experience of Liaw and Wiener [27], this estimation tends to be quite accurate. They do note that research by Bylander has shown that the OOB can have a tendency to bias upwards. The benefit of using OOB to calculate the error rate is that there is no need for an extra set of testing data.

2.2.4.4 Variable importance

It is not known yet what the correct sensor placement is to measure posture, nor which data is necessary. One way to test this is to start off with a large number of sensors and to determine which of these sensors can be eliminated without affecting the quality of the output too much. Breiman developed four methods to measure the importance of different values. He explains these in his "*Manual on setting up, using, and understanding random forests V3.1"* [26]

Method one: The first method starts by randomizing the decision notes determined belonging to the value that is being tested, and then comparing the internal error rate. The randomization is done by identifying the decision notes that rely on the variable in question, and making their outcome random. Then internal error rate of this new adjusted forest is compared with the error rate of the original forest. A small increase in the internal error rate indicates a not so important variable, while a large increase in the internal error rate indicates a variable of high importance.

Method two: The second method starts in the same way as method one, by randomizing the notes reliant on the relevant variable. Instead of comparing the internal error rate, this method compares the margin of the output of the new forest to that of the old forest. This is done for one specific case in the data.

Method three: The third method is quite similar to the second one, but instead of comparing the margins of one case directly, this method compares the number of margins gone up versus the number of margins gone down over all test cases.

Method four: The fourth method uses the impact of a variable on the Gini index to estimate its importance. By adding up the total decrease in the Gini index caused by the variable in question, one should get a reasonable idea of the importance of this variable. This number is normalized by the number of trees. A variable that has a lot of influence on the Gini index should theoretically be of higher importance than one that does not.

2.2.4.5 conclusion random forests

In short, random forests are a type of assemble learning technique that makes use of many decision trees. Each tree is constructed using a random subset of the training data and each node in the tree uses only a few random variables for its split. This randomness gives random forests an advantage over normal decision trees as it helps prevent the overfitting of the model.

One big advantage random forests have over other machine learning techniques is their ability to estimate variable importance. Originally, four methods for variable importance estimation where developed. Three of these methods depend on randomizing the decision nodes that are dependent on a specific variable. By testing the model before and after the randomization, the usefulness of a specific variable can be compared to a random 50/50 split. The fourth method calculated the impact a variable has on the Gini index. In theory, a variable that causes a large total decrease in the Gini index should be more important than a variable that has a smaller impact.

2.3 Similar products

There are various products on the market that promise the user real time feedback on the quality of their posture. Two of these are Lumo lift[36] and Upright Go[37]. Both these "smart wearables" use a single sensor placed on the body for their measurements. Lumo Lift uses a clip-on sensor that can be clipped onto the clothing of the wearer. The Go uses a sensor that has to be placed directly on the back of the user. The data that can be gathered from the single sensor module is limited. This is noticeable in the performance of the Lumo Lift and the Go. Visser[11] found in personal tests that the Lumo Lift missed certain bad postures. For example, an incorrect position of the scapulae was not noticed is the rest of the posture was correct.

The sensor in the Lumo Lift can also be used for other measurements. The application provided with the device can estimate the number of steps taken, the estimated distance walked, and the estimated calories burned. This turns the device in a more general fitness tracker instead of just a posture measuring device.

As far as wearable posture sensors go, most of the products on the market seem to take a similar approach as the Lumo Lift and the Upright GO. For example, the Opter Pose[38] is a Kickstarter project proposing a sensor on a necklace. This sensor seems to work similarly to the GO and the Lift. One different way to measure posture is via the seat rest. The Darma[39] is an example of such a device. Of course, this type of device is not wearable. Finding out the accuracy of these posture measuring devices is hard because the companies that produce them do not supply any studies on the effectiveness of their devices. Because of this it is hard to say which device works best.

2.4 Conclusion

Defining the difference between good and bad posture is a complicated problem. Based on a review of the literature, an interview with a physiotherapist done by another bachelor student, and an interview with an orthopaedic surgeon, it was possible to create a general definition sufficient for this project: good posture is posture that allows for efficient use of the muscles required to maintain said posture while avoiding unnecessary stress in the vertebrae and joints, bad posture is posture that is not good posture.

Various machine learning techniques were evaluated for posture measurement. Eventually, Random Forests was determined to be the most promising technique for its ability to estimate variable importance. The variable importance measures of the random forest algorithm could help reduce the number of sensors needed for posture measurement.

3 Ideation

3.1 Sensors

In this part of the report possible sensors and sensor positions will be discussed. First, there will be a diverging part to identify as many possible sensors as possible. Then, there will be a part looking at the xSens MVM suit that incorporates multiple sensors in compact sensor units. Lastly, there will be a diverging part to determine that sensors that will actually be used.

3.1.1 Sensors for each type of bad posture

This part of the report is part of the diverging process. Ideally each bad posture in Figure 6 to Figure 8 should be measurable. To get an overview of what kind of sensor might be helpful, or even needed, a seemingly suitable sensor and sensors placement for each type of bad posture was determined.

The bad postures in the sagittal plane are a combination of the position of the pelvis, head, and the shape of the spine. To measure these parts of the body in relation to each other, gyroscopes could be placed on the pelvis, along the spine, and on the head, see Figure 9. This would allow for a comparison between the angles of these different parts of the body, allowing for detection of a correct shape of the spine. Another way to keep track angles between these parts of the body is by using stretch and bend sensors. For example, stretch sensors could be placed on the neck to detect a down- or upwards bend. The stretch sensor on the back of the neck could even be extended to cover the upper back. A bend sensor could be placed underneath the sternum to help detect a slouching posture.

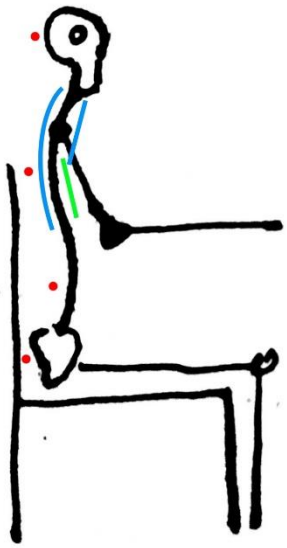


Figure 9, posture sensors in the sagittal plane. Red indicates a gyroscope, Green indicates a bend sensor, and blue indicates a stretch sensor.

Sensing Neck-position: There are also more unconventional ways to try and detect posture, specifically when talking about the position of the head and neck. Not having to place a weird sensor on top of the head would help with the wearability of the device. However, it also makes sensing head position more of a challenge. On idea would be to attach proximity sensors to the trunk of the user. These sensors could detect the head movements above them. It is likely that these sensors would be prone to error due to small changes in their positioning, making them hard to use. Another idea would be to attach capacitive sensors close to the neck. These sensors could detect the neck bending closer to

them, which would give an indication of the position of the neck. Lastly, the ears could be used as an appropriate place for sensor placement. Firstly, sensors can be attached from the earlobes like jewellery. Secondly, they can be clipped on the ear similar to some hearing aids and earphones. Lastly, they can be placed behind the ear, hiding them form view. The most obvious sensor to place at the ear would be a gyroscope. However other interesting options are possible. One idea would be to attach a magnet to the earlobe, and a hall sensor to the shoulder, allowing for the position of the ear over the head to be measured. Another idea is to attach stretch sensors from the earlobe to the shoulder. A similar idea to that of the hall sensor would be to attach a photodiode to the ear and an LED to the shoulder. See Figure 10 for more information about the placement of the sensors discussed.

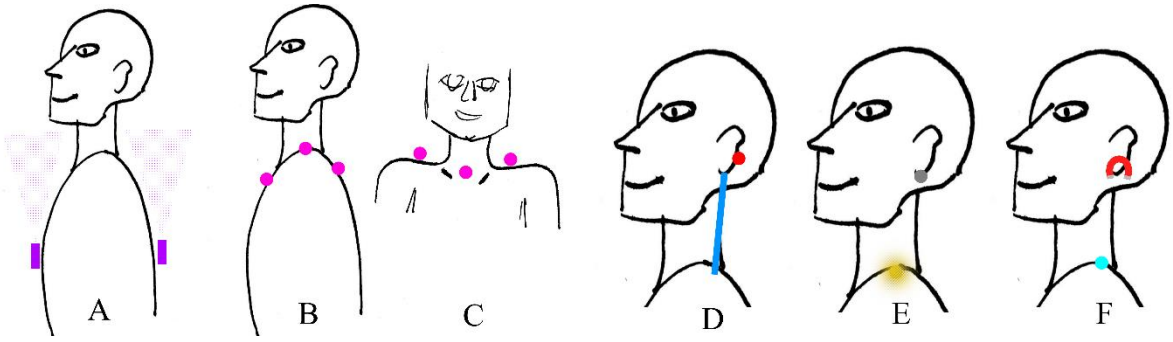


Figure 10, Sensors measuring the position of the neck. A: proximity sensors on trunk. B, C: capacitive sensors near neck. D: gyroscope (red) on ear and stretch sensor from ear lobe to shoulder. E: LED with photodiode. F: magnet with Hall sensor.

Similarly, to the sagittal plane, bad postures in the coronal plane can be the result of an incorrect position of the pelvis, head or spine. Additionally, bad posture in the coronal plane can be caused by incorrect positioning of the shoulders. The sensors discussed in the previous section about the sagittal plane and the neck can also be useful for posture measurement in the coronal plane. Seemingly the only thing in the coronal plane that could not be measured this way is the position of the shoulders. To make detection of the shoulder position possible it is likely that extra sensors are needed. On example would be adding gyroscopes to the shoulder. Another option would be to attach a stretch sensor under the armpit to detect the shape of the muscles reacting to the position of the shoulder. Instead of a stretch sensor under the armpit, a pressure sensor might work for similar measurements. Lastly, adding a stretch or bend sensor horizontally along the shoulders could be useful. This sensor could be attached to both shoulders, but it would also be possible to attach this sensor to the scapulae. As discussed earlier, the scapulae should be straight when maintaining a correct posture. When incorrect posture is adopted the scapulae move away from each other, causing them to pull on the stretch sensor. See Figure 11 for more information on the sensor placement that was just discussed.

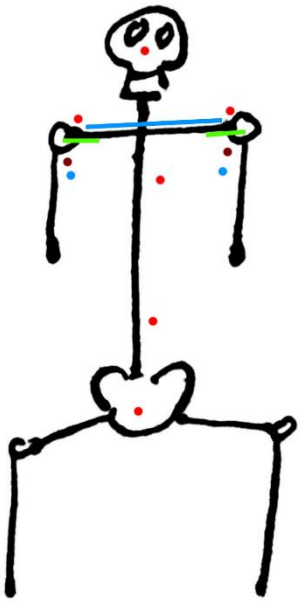


Figure 11, posture sensors in the coronal plane. Red indicates a gyroscope, Green is a bend sensor, blue indicates a stretch sensor, and brown indicates a pressure sensor.

The last plane to discuss is the transverse plane. Posture in the transverse plane is determined by position of the shoulders and the rotation of the spine. Using the gyroscopes discussed before, it should be possible to measure these things. Specifically, a gyroscope on the lower back and one on the upper back could be useful, see Figure 12.

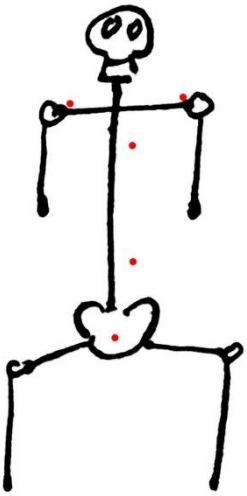


Figure 12, posture sensors in the transverse plane. Red indicates a gyroscope.

One type of sensor that has not been mentioned yet is the accelerometer. Accelerometers are commonly used on body sensors. There are various potential ways to use accelerometer measurements to try to determine sitting posture. One way would be to replace all gyroscopes discussed previously with accelerometers in the hope that the accelerometer data will be able to fulfil the same function. Another idea would be to only use the pelvis and the head placement in an attempt to track the distance between these two body parts. This type of accelerometer use probably needs some type of calibration of the upright posture and some way to keep track of the movement history as accelerometers say nothing about their current position, they only measure acceleration.

One seemingly more unconventional way to measure posture quality would be to attach one or more accelerometers to the users' shoulder or neck. It might be possible to predict the user's posture based on the small corrections made by the body to keep itself upright. These small corrections might change when one's posture is changed.

3.1.2 xSens

The xSens MVM Analyze[40] is a product meant for real time capture of movement data. The suit is meant to be usable outside of laboratory conditions. The suit comes with a base unit and various sensor units. It was designed to be an affordable, full body, motion capture suit. The matching software allows for the calculations of joint angles and the centre of mass of the user. Each sensor unit contains a 3D accelerometer, 3D gyroscope, 3D magnetometer (that measures the earth's magnetic field), and a barometer. The sensor units are placed on nearly all parts of the body, including the hands, feet, and head. They can be attached using either straps or the Lycra suit, see Figure 13.

Figure 13, xSens MVM Analyze suits. A: Lycra suit. B: Straps [41]

During this project, it might be possible to use the xSens MVM Analyze suit to for measurements. The onboard gyroscopes and accelerometers could be suitable for the types of measurements described in

section 2.2.1. The placement of the sensor on the xSens suit is a little bit different however, making it potentially inconvenient to work with.

3.1.3 Types of sensors that will be used

In part 3.1.1 of this report an overview was made of different sensors that might be useful for posture measurement. Not all of these sensors will be used during the tests. Out of all the sensors listed, the most promising few where chosen. These sensors are: the gyroscope, the accelerometer, and the stretch sensor.

The gyroscope was chosen as known to be a reliable sensor for full body tracking. The xSens suit mentioned earlier is an example of a full body tracking suit that uses gyroscopes as part of its measurements. Gyroscopes are also compact. Making them easy to use as wearable sensors. Lastly, gyroscopes measure angles, which seems like an ideal type of measurement for posture recognition as the angles between different parts of the body determine the quality of the posture. The gyroscopes will be placed in a way that allows for a comparison of the angles between the different parts of the body that are important for a correct sitting position. Such positioning of the gyroscopes has been determined in part 3.1.1, see Figure 14. This involves placement on the pelvis, lumbar curve, thoracic curve, shoulders, and head. The sensor placed on the lumbar will be positioned in the "deepest" part of the curve to try to get an accurate measurement of the state of the lumbar. The gyroscope placed on the thoracic spine will be placed slightly higher up in an attempt to pick up more of the neck movement.

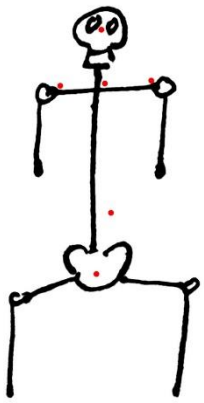


Figure 14, red dots indicate the position of a gyroscope.

Just like gyroscopes, accelerometers are often used for full body tracking. In fact, most gyroscopes sensors modules come equipped with an accelerometer as well. Because of this it is relatively simple to not only include accelerometers in the test as well as gyroscopes.

Stretch and bend sensors seem like an interesting alternative way to measure the position of parts of the body relative to each other. Benefits of stretch and bend sensors are that they are relatively cheap and easy in interface. However, a disadvantage of these kind of sensors is that they can be hard to get useful measurements out of. Stretch and bend sensors need to be positioned tightly to the body in order to make them work properly. This makes setting up the sensors a tricky task, especially when people try to do it by themselves. Stretch and bend sensors also have a reputation to be quite fragile, requiring extra care from the user. For this research it was chosen to only focus on stretch sensors, instead of using both stretch and bend sensors. The reason for this is that the available stretch sensors have a reputation to be more reliable than the available bend sensors. Theoretically, bend and stretch sensors can perform the same task by placing them on the opposite side of the body, allowing for all

the same types measurements to be done using only stretch sensors instead of both stretch and bend sensors. For the final position of the stretch sensors see Figure 15.

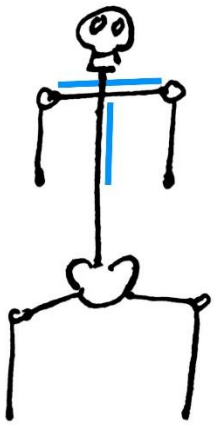


Figure 15, blue lines indicate the position of a stretch sensor

In chapter 2.2 many more different kinds of sensors are mentioned. Most of these sensors will not be used during the actual tests. Especially many head and neck position sensors are left out of the tests. This is because they are expected to either not work very well, or to stretch the limits of what can be considered wearable. The xSens suit is also left out of the tests. This is because the sensor placement of this suit is not expected to be ideal for posture measurement. The sensors used in the suit remain promising, namely the gyroscope and the accelerometer. However, instead of using the xSens suit to implement these sensors, they will be implemented manually.

4 Specification

In chapter 2 the theoretical background of the research was given. This chapter will focus on creating a concrete plan to obtain and analyse the desired data. To do this it will outline what sensors will be used, what machine learning techniques will be used, the way the suit will be constructed and the way the test will be conducted.

4.1 Sensors

4.1.1 Sensor modules that will be used

To implement the gyroscope and the accelerometer, it is most convenient to use a sensor module that combine both. This makes it easier to connect the sensors and it makes sure all the axis line up correctly. There are various modules that provide this functionality. One of these modules is the MPU6050 [42]. The MPU6050 is generally considered to be one of the more precise and reliable gyroscope and accelerometer modules available in the consumer market. Because it is a consumer product, it is widely available and affordable. The MPU 6050 uses a I²C bus the communicate. This allows for reliable communication with an Arduino. Because the I²C bus of each sensor module tries to write to the same I²C address, it is not possible to easily control all 6 MPU6050 modules at the same time. To get over the limitations of the MPU6050 I²C implementation an Adafruit TCA9548A I²C [43] multiplexer will be used.

The stretch sensors that will be used are Adafruit conductive rubber cords[44]. These elastic cords increase in resistance when they are stretched. Integrating one of these cords into a voltage divider allows for an Arduino to measure the amount which the cord is stretched out.

4.2 Suit

The suit housing the sensors will need to provide suitable sensor placement on many different people with different kinds of bodies. It must be able to place the sensors on the correct part of the body, while also holding them tight enough to allow for accurate and consistent measurements. To ensure a properly working suit, the design process will occur in multiple stages. First, a basic suit will be made. This suit will then be tested on 2 to 3 people. The experience gathered during these tests can then be used to improve the suit for the full-scale measuring sessions. In the end, multiple full-scale measuring sessions were held. The suit was changed based on the experience of each new session.

4.2.1 First version sensor suit (used in session one and two)

In order to make the suit as adaptable as possible, a combination of stretchy fabric, elastic bands and Velcro is used. The suit is be based on a stretchy t-shirt. The sleeves are removed of this shirt because they are not necessary for the sensor placement and they might get in the way of the wear ability of the suit. To allow the suit to adapt to different body types, the side seams of the t-shirt are ripped open. These seams can then be closed using elastic bands and safety pins. This allows for the shirt to be fitted exactly to the wearer's body.

Because every body is different, the sensor position on the suit needs to vary a bit from person to person. To allow for easy repositioning of the sensors, velcro is used. Each sensor is attached to a little strip of velcro that can then be attached to a velcro strip on the suit. This allows for easy and fairly precise adjustment of the sensor placement on the suit to match different bodies. See Figure 16 or the placement of the velcro strips on the suit and the corresponding sensor position. A picture of the real sensor suit can be found in appendix F.

The sensors will be interfaced via an Arduino Nano[45], which is ideal due to its small form factor. The Arduino will use Serial communication to communicate the sensor values to a pc, where they can be stored and processed. The Arduino Nano and the Adafruit multiplexer used for the I²C communication will be sown onto the suit to keep them safe and stable. The Arduino code can be found in appendix B. The code is based example code on how to implement the MPU 6050 by Jeff Roberg[46]. An Adafruit tutorial[47] on how to implement their TCA9548A multiplexer was also used. The Arduino code processes the data received from the sensors into roll, pitch, yaw angles for

the gyroscopes and a x, y, and z acceleration for the accelerometers. The accelerometer angles are relative to the “world”. This means that the gyroscope angles are used to compensate for the rotation of the accelerometers, allowing the x, y and z dimensions to remain in the same relative orientation relative to the world at all times. Apart from this, no special processing of the data is done.

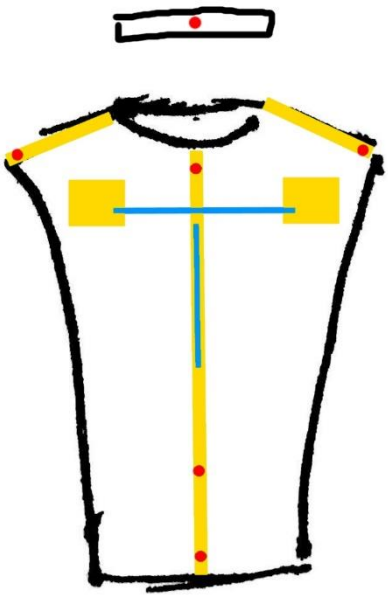


Figure 16, velcro and sensor layout on sensor suit. yellow indicates velcro strips. Blue indicates stretch sensors. Red indicates gyroscope/accelerometer modules.

4.2.2 Second version sensor suit (used in session three)

The second version of the sensor suit does not differ much from the first. The physical side is completely the same. The only change comes in the form of code. The first version of the code sends the latest sensor data available to the computer, without any regards of the sensor values that came before it. Because of this, random noise in the sensor values can have a large impact on the data. In the second suit however, the data that is communicated to the computer is an average of the sensor values over a 1 second interval. This averages out the random noise and should produce cleaner data.

4.2.3 Third version sensor suit (used in session four)

The third version of the sensor suit is mostly the same as the previous versions. The Physical suit is exactly the same as before. The only change comes in the form which the accelerometer data is processed. In the previous measuring sessions, the accelerometer data was recorded in Cartesian coordinates. The movement over each of these coordinates was averaged over the moving window, this number was then used to train the machine learning algorithm. In the new system, spherical coordinates are used. These spherical are all normalized to have a length of 1. Meaning that only the angles of the vector are important, and not its size. The average angles are taken over a one second window and used to train the machine learning model. The goal of this new way of storing the accelerometer data is to focus more of the angle of the acceleration vectors instead of their magnitude.

4.3 Machine learning

4.3.1 Data processing

The values outputted by the sensors are only really relevant when they can be related to something. This is because the values outputted by the sensors are relative to their own position. For example, a gyroscope indicating an angle of 95 degrees does not mean anything unless you can compare it to some sort of baseline angle. Because of this, it is expected that all sensor data will need to be related to some kind of baseline. One way to process the data is by subtracting a line of correct posture data from it. This way the data shows the difference between the current posture and the correct posture. Another way to relate sensor data to each other is by comparing the different sensors against each other. For example, the difference between the angles of two gyroscopes. Using the different sensors without any kind of baseline will likely result in data that is limited in its use.

4.3.2 Machine learning software

The machine learning model will be trained using the free machine learning library from Scikit Lean[31]. Scikit learn allows for easy implementation of various machine learning algorithms, including the random forest (RF) algorithm. RF is the main machine learning algorithm that will be used as it is able to both perform classification tasks and variable importance estimations. See chapter 2 for more information about different machine learning techniques. One benefit of Scikit learn is that it makes it easy to switch between different machine learning techniques. This allows for experimentation with different techniques and their accuracy. The python code can be found in appendix A. This code is based on a tutorial by stackabuse.com[48].

4.4 measurement design

The measurement data needs to be gathered in a systematic way to ensure consistent measurement conditions and representative data. For this reason, precise measuring procedures are described in this chapter.

First, the test subject will be asked to read through and sign the consent form and matching information sheet. Any questions asked by the participant need to be properly answered. When the consent form has been signed by both the participant and the researcher, the sensor suit needs to be placed on the test subject. This needs to be done by the researcher. The researched need to ensure the suit is placed tightly around the body of the test subject and that all sensors are positioned correctly, taking extra care that the stretch sensors are being activated when they should.

The actual data will be gathered on the basis of the good and bad postures identified in chapter 2, see Figure 6 till 8. The participant will be asked to sit in all different positions identified in these figures. For each of the postures one or more lines of data will be gathered. During the measuring sessions the Arduino of the sensor suit will be connected to a laptop via USB. To prevent any harm done to the participant by the electronics, the laptop cannot be connected to the power net via a charging cable. Afterwards, the participant will be relieved of the sensor suit and thanked for their cooperation.

4.5 Consent form and ethical approval

Because the research will involve the use of human participants, it is necessary to follow certain ethical guidelines. Firstly, ethical approval needs to be had from the university of Twente ethics commission. The necessary information form can be found in D. The research was approved by the ethics commission and was assigned reference number: RP 2019-37. Secondly, it is necessary to have every participant read and sign a consent form an accompanying information sheet. This assures that they are aware of the research they are participating in and that they are aware of their rights as a participant. The consent form and information sheet can be found in appendix D and E. Both the consent form and the information sheet where checked by the ethical commission in order to obtain ethical approval.

5 Realisation.

This chapter aims to give an overview of the different iterations of the data gathering sessions and sensor suit. It discusses the changes made in each iteration and the results that were obtained during the data analysis. The suit and measurement design are based on the ones described in the specification chapter. Each iteration of the measurement cycle adds the relevant changes to this design. For the detailed information about the data analysis and results, see chapter 6 “Data Analysis”.

5.1 Pilot measuring session.

5.1.1 Result from the pilot measuring session.

In order to identify any problems with the suit, code, and data analysis, a small pilot study was conducted. This study collected the data from 3 different test subjects. These sessions went well, with the test subjects being able to understand the things they were supposed to do. The suit also did not constrain them too much, allowing them to perform the desired movements without any inhabitations. To increase the amount of data gathered, multiple “lines” of data where gathered for each sitting position. This data was gathered in quick succession, not allowing for much movement in between each line of data. This resulted in data that was very similar to each other, possibly misrepresenting the accuracy of the machine learning model.

The suit performed well most of the time, but every now and then the Arduino program would crash, requiring a restart of the device. The suit had a bigger problem with the stretch sensors not working as expected. Especially the sensor running across the shoulders did not activate as much as expected.

During the data analysis it became clear that the data was not outputted in an efficient format, requiring significant alterations before being used to train the machine learning model. Using the raw data, the random forest algorithm had troubles creating an accurate model. This was likely caused by variations in gyroscope output every time they were reset, causing the data from different measurements to be almost useless in comparison to each other. To improve the predictive abilities of the model, the data was processed by relating it to data from the correct sitting position. In practice this meant that the sensor values from the correct sitting position where subtracted from all the other values. Making all data relative to the data from the correct posture. This greatly improved the accuracy of the model, making it hit 100 percent accuracy in many situations.

5.1.2 Improvements made based on the pilot measuring session.

To solve most of the problems identified during the pilot session, the suit, code, and way of doing measurements was changed slightly. First of all, a slight alteration in the way the Arduino outputted the data was made. The goal of this was to make the manual data processing less intensive. Secondly, extra elastic straps where added to the suit in an attempt to improve the output of the stretch sensors. Most notably, an elastic band underneath the armpit was added to further tighten the shoulder stretch sensor to the body of the test subject. The measuring protocol was changed as well, instead of gathering multiple lines of data in quick succession for each sitting position, each test subject will be asked to go through all the sitting positions multiple times. This should increase more variance in the data, making it more representative.

The cause of the suit crashing was identified to be a short circuit between a stretch sensor and an MPU 6050 module. By slightly moving the position of these sensors, the crashing was significantly reduced. The crashes where not completely gone however, but infrequent enough to not cause significant problems. After each crash new baseline needs to be gathered. The new measurements then need to be processed with this new data.

5.2 First measuring session.

After completing the pilot study, the first “full scale” measuring session was conducted. The purpose of this session was to gather enough data to be able to come to useful conclusions using the machine learning model. During this measuring session 9 different people where measured. Each person took

on each posture in figure 6 till 8 twice to increase the amount of data available. During these measurements, the improvements designed based on the results from the pilot study where implemented. This resulted in better readings from the stretch sensors, and more variance in the data.

5.2.1 Improvements based on the first measuring session.

During the data analysis it was discovered that a mistake was made in the way the data was gathered. When processing the data, one line of correct posture data is needed to act as a baseline as a baseline. This line of correct posture data can then not be used to train the machine learning model anymore. This resulted in a significant lack of correct posture data. The lack of correct posture data was aggravated by the way the measurements were conducted. Because figure 6 to 8 only contain one "kind" of correct posture and many more kinds of incorrect posture, a disproportionate amount of incorrect posture data was collected in the first place. The lack of correct posture data was visible in the results of the data analysis (see chapter "data analysis"). Most of the wrong guesses made by the random forest model concerned the correct posture data.

5.3 Second measuring session.

The goal of the second measuring session was to expand upon the data from the first sessions. Based on the results of the first session, it was concluded that more correct posture data is needed in order to improve the accuracy of the machine learning model. To do this, three extra test subjects where tested. These sessions focussed mainly on gathering correct posture data, while also adding a bit to the incorrect posture data. The data from the first session was combined with the data from the second session to create an extra-large data set with data from 12 different people.

5.3.1 Improvements based on the second measuring session.

During the data analysis, two extra problems with the data were found. Firstly, almost all of the correct posture data came from only a few test subjects, while most of the incorrect posture data came from the others. This resulted in an unnatural split between the correct posture data and the incorrect posture data. Secondly, there was a problem with the way the data was recorded. The data recorded by the sensor suit was based on a single sensor value, without any processing done to it. resulting in a big influence of random noise and movements on the data. This can be improved by taking an average of the data over a set amount of time.

5.4 Third measuring session.

The third measuring session was designed to improve upon the mistakes made in the first and second session. Firstly, the second session had a division in the data. With most correct posture data coming from the people measured in the second session and most incorrect posture data coming from the first session. This could mean that the data is unrepresentative. To avoid similar problems during this measuring session, an almost equal amount of correct and incorrect posture data was gathered from each test subject in this test.

The second problem with the first and second sessions relates to the way the sensor data was read and stored. The measuring suit in the earlier test recorded the latest values from the sensors without any processing done to them. This approach is very sensitive to random noise and small movements. To make the sensor values more stable, the sensor suit in the third session took the average sensor values over one second.

5.4.1 results from the third measuring session.

When training the random forest model using the original, unprocessed, data an accuracy of 86 per cent was achieved. In an attempt to improve the accuracy of the model, the data was processed using correct posture as a baseline (see 4.3.1 Data processing). This increased thee accuracy to 92 per cent. Another way of processing the data is by relating it to a specific sensor. When relating the data to the upper back sensor an accuracy of 80 per cent was achieved, not nearly as good as the other methods.

The random forest variable importance measures indicated that both shoulder sensor modules

and the sensor module on the head band where the most important. When training the model using only the data from these modules an accuracy of 85 per cent was attained. Using only the left shoulder sensor instead of both the left and right did not decrease this accuracy. Even using the left shoulder module without the module on the head band did not affect accuracy at all, maintaining an accuracy of 85 per cent.

In real life it is common for people to move throughout the day. For example, when sitting next to a curved desk someone might rotate on their chair to use different parts of said desk. To simulate the effect this might have on a machine learning model, this angle of rotation (the pitch angle) was removed from the gyroscope data. Training the machine learning model using the data without the pitch angle resulted in an accuracy of 84 per cent. For an overview of the accuracies of the different machine learning models that where discussed during this session, please see Table 1. For a more detailed analysis of the data see the next chapter, "Data Analysis".

Dataset:	**Accuracy of the random forest model (%):**
Unprocessed data.	86
Data relative to correct posture.	92
Data relative to the upper back modules.	80
Only using the head and both shoulder modules, data relative to correct posture.	85
Only using the head and left shoulder modules, data relative to correct posture.	85
Only using the left shoulder module, data relative to correct posture.	85
Data without pitch angle, relative to correct posture. (all sensors)	84

Table 1, the accuracy of various models trained using different datasets. All data from 3rd measuring session.

5.4.2 improvements based on the third measuring session.

One improvement could be made on the way the data was gathered during this session. The current measuring setup does not take into account that the users of a posture measuring device might move around. For example, the user might sit in an office chair that is able to rotate. This rotation could throw off the measuring device. For further measurements, this rotation should be taken into account.

5.5 Fourth measuring session.

The fourth measurement session was designed not only to improve on the third session while also testing a different approach to the way the data is stored. Two big changes were made compared to the third session. Firstly, this session included the type random position changes that might occur when working at a desk. For example, a user might rotate their chair to access different parts of their desk while they are working or move to a different desk entirely. These changes were simulated by asking the test subject to sit on an office chair and having them rotate in random directions between each sitting position. During the session the test subjects rotated in a 180-degree window.

The second way in which the session differs is in the way it records the accelerometer data. In the previous measuring sessions, the accelerometer data was recorded in Cartesian coordinates. The movement over each of these coordinates was averaged over the moving window, this number was then used to train the machine learning algorithm. In the new system, spherical coordinates are used. These spherical coordinates all have a length on 1. Meaning that only the angles of the vector are important, and not its size. The average angles are taken over a one second window and used to train the machine learning model. The goal of this new way of storing the accelerometer data is to focus more of the angle of the acceleration vectors instead of their magnitude.

During the fourth measuring session, 7 different people where measured. Each person went through the measurement process twice to increase the total amount of data.

5.5.1 results from the second measuring session.

Training the random forest model using unprocessed data resulted in an accuracy of 75 per cent. This is much lower than the 86 per cent using the data from the third session. After processing the data to make it relative to the correct posture, the accuracy increased to 84 per cent. Again. This is lower than the 92 per cent achieved during the previous session. Interestingly, this is the same accuracy as the model in the previous sessions that had all data from the pitch angle of the gyroscopes removed. that model was designed to simulate the measuring sessions of this session.

The random forest variable importance measures gave a high score to the gyroscopes on both shoulders, the headband, and the lower back. This is similar to the results of the previous analysis, except for the added importance to the lower back gyroscopes. The spherical coordinates based of the accelerometer data where estimated to be fairly unimportant by the variable importance measures. In an attempt to reduce the complexity of the sensor suit, the random forest model was trained using only the data from the sensor modules on the shoulders, head, and lower back. This model had an accuracy of 83 per cent. Removing the right shoulder from the dataset resulted in an accuracy of 81 per cent. Training this model without the lower back sensor module, so using only the left shoulder and head band, did not result in a decrease in accuracy, retaining the 81 per cent accuracy. Training the model using only the data from the head band and the lower back gave an accuracy of 79 per cent.

Interestingly, the variable importance estimations greatly preferred the gyroscopes over the accelerometers. When training the model using only accelerometer data or only gyroscopes data, these predictions are conformed. The model using only gyroscopes achieves an accuracy of 84 per cent. The same as the model using all sensor modules. The model using only accelerometer data achieves the abysmal accuracy of 39 per cent, worse than would be expected from a random 50/50 guess. Training the model using stretch sensors only results in an accuracy of 56 per cent. Again, confirming that the gyroscope data is by far the most useful for posture detection. For an overview of the accuracies of the different machine learning models that where discussed during this session, please see Table 2. For a more detailed analysis of the data see the next chapter, “Data Analysis”.

Dataset:	**Accuracy of the random forest model (%):**
Unprocessed data.	75
Data relative to correct posture.	84
Only using the head, lower back, and both shoulder modules, data relative to correct posture.	83
Only using the head, lower back, and left shoulder modules, data relative to correct posture.	81
Only using the left shoulder and head module, data relative to correct posture.	81
Using only the lower back and head module, data relative to correct posture.	79
Using only gyroscopes, data relative to correct posture.	84
Using only accelerometers, data relative to correct posture.	39
Using only stretch sensors, data relative to correct posture.	56

Table 2, the accuracy of various models trained using different datasets. All data from 3rd measuring session.

6 Data Analysis

6.1 Data analysis session one

During the data analysis it was discovered that a mistake was made in the way the data was gathered. Because all the data needed to be processed using the straight posture data, there was no straight posture data left over to train the machine learning model. To solve this problem, extra data had to be gathered. This extra data gathering focussed mainly on gathering straight posture data. Because of a lack of time, the data was gathered by placing the sensor suit on myself. After obtaining the extra correct posture data, and some extra incorrect posture data, the data could be processed properly. However, this meant that the correct posture data available to train the model was relatively limited and all based on one person.

6.1.1 Training the random forest model using unprocessed data

To test the need for the baseline correct posture data the random forest model was first trained using the "raw", original, data. This data does not contain the extra data gathered from the researcher himself. Eighty percent of the data was used to train the model, while the other 20 percent was used to test the model. A model was produced using both twenty estimators and one-hundred estimators to test the difference in accuracy. This model was tested using five different training and testing splits. These training and testing splits where generated using the Scikit learn "train_test_split" function. The average accuracy of the 20-sample model and the 100-sample model was virtually the same, with the 20-sample model having an accuracy of 0.90000 out of 1 and the 100-sample model having an accuracy of 0.90625. See Figure 17 for the accuracies for each testing/training split.

Figure 17, the accuracy of the random forest model using different training and testing splits, "raw" data.

All Figure 20 does show which predictions where wrong and which were right. A confusion matrix shows the spread of the predictions. Adding the confusion matrices from all training splits together results in the confusion matrix in Figure 18. This shows that most incorrect predictions are wrongly classified correct posture data, with only one incorrect prediction being a misclassified as correct posture. This is most likely due to the difference between the amount of data gathered on wrong posture versus the amount of data gathered on correct postures. This difference resulted from gathering data for each of the postures identified in Figure 5 to 6. These figures contain more bad postures then good postures.

	wrong	correct
wrong	140	1
correct	15	4

Figure 18, all confusion matrices of the 20 tree models combined, unprocessed data. Green indicates the real class the data belongs to, yellow the predicted class.

6.1.2 Training the random forest model using data relative to the correct posture

As described before, processing the data required gathering extra correct posture data. This data was gathered by placing the sensor suit on the researcher instead of using independent test subjects. This means that all the correct posture data is from the same person, making it less representative of people's posture as a whole.

As expected, using posture data that is relative to the correct posture results in a significantly higher accuracy of the predictions. Just like before, both the 20-sample model and the 100-sample model have similar average accuracies. With the 20-sample model having an accuracy of 0,985 and the 100-sample model having an accuracy of 0,975. Figure 19 shows the accuracies of each testing/training split. Two out of the five training testing splits reached an accuracy of 1.0, indicating that they were one-hundred percent accurate.

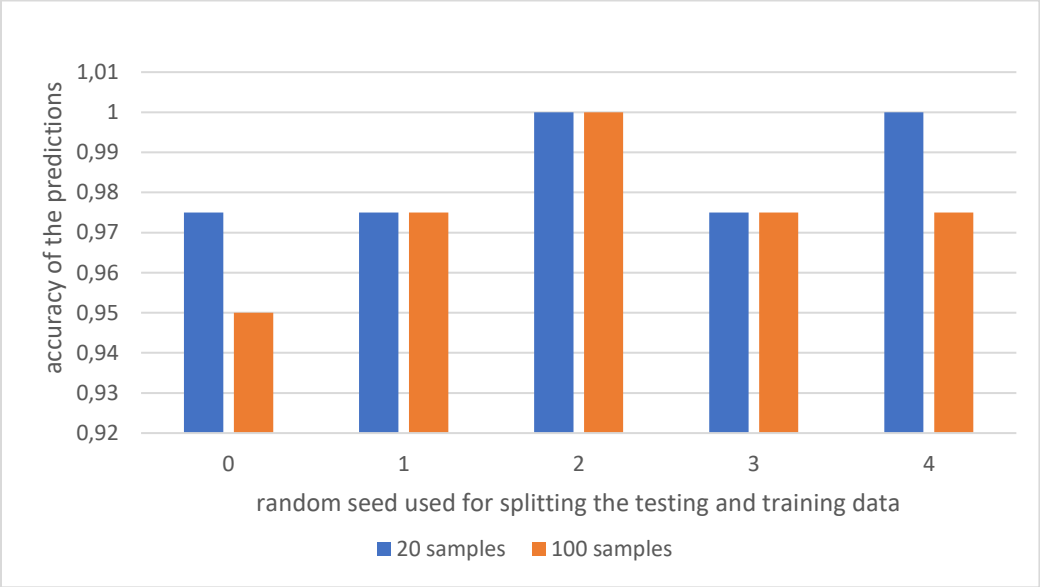


Figure 19, the accuracy of the random forest model using different training and testing splits, data relative to correct posture.

	wrong	correct
wrong	189	0
correct	3	8

Figure 20, all confusion matrices of the 20 tree models added up, data relative to correct posture. Green indicates the real class the data belongs to, yellow the predicted class.

Similarly, to the unprocessed data, most of the wrong predictions made while using the processed data are correct postures labelled as incorrect. Indicating a need for more correct posture data.

6.1.3 Analysing variable importance.

The build in variable importance measures of the random forest algorithm can give an idea of what sensors are the most important. Figure 21 shows the top five most important variables for the train/test splits that produce the most accurate models using 20 trees. All data used to create the models is relative to the correct posture.

Seed random split	4	2	3	0
Most important variable	11	11	10	11
Second most important variable	10	3	11	9
Third most important variable	9	4	16	10
Fourth most important variable	24	16	9	24
Fifth most important variable	15	10	3	4

Figure 21, top 5 most important variables for different train test splits.

Especially variable 11, 10, and 9 seem important according to Figure 21. These variables correspond to the three axes of the accelerometer on the right shoulder. When training the random forest model using only the data from this sensor, an accuracy of 0,97 is achieved. See Figure 22 for the corresponding confusion matrix. Figure 25 shows that a lot of mistakes are made when identifying correct posture. The confusion matrix seems to have a similar distribution to the one using all sensors, but with a lower accuracy.

In retrospect, the importance of the right shoulder's data seems to make sense. Movement of the upper body and arms is almost always going to have an effect on the position of the position of the shoulder, allowing for the shoulder position to be an important indicator of sitting position. This explanation, however, would imply that the left shoulder's variables should be just about as important. One would also expect to see a higher importance of the gyroscopes attached to both shoulders. Of course, the variable importance measures are just estimating the real importance of the variables and, as discussed in chapter 2.3.2, they are not always accurate.

	wrong	correct
wrong	189	0
correct	6	5

Figure 22, all confusion matrices of the 20 tree models combined, data relative to the correct posture, only data from the accelerometer on the right shoulder. Green indicates the real class the data belongs to, yellow the predicted class.

When training the model using only the gyroscope data from the right shoulder, the predictive ability of the model becomes near zero. Figure 23 Shows the confusion matrices of the models added up. Only one of the correct postures was correctly categorized as such. One might argue that the low predictive capabilities of the model are partly caused by the lack of "correct posture" data. But that does not explain why the model using the accelerometer does so much better.

	wrong	correct
wrong	189	0
correct	10	1

Figure 23, all confusion matrices of the 20 tree models combined, data relative to the correct posture, only data from the gyroscope on the right shoulder. Green indicates the real class the data belongs to, yellow the predicted class.

To test, and compare, the predictive capabilities of the accelerometer on the left shoulder compared to the right shoulder, a random forest model was trained using the data from this sensor as well, see Figure 24.

	wrong	correct
wrong	189	0
correct	11	0

Figure 24, all confusion matrices of the 20 tree models combined, data relative to the correct posture, only data from the accelerometer on the left shoulder. Green indicates the real class the data belongs to, yellow the predicted class.

The data in figure 27 shows that the gyroscope on the left shoulder performs just as badly as the accelerometer on the right shoulder. In an attempt to further understand the difference between the right shoulder and left shoulder data, a closer look was taken the data. This showed that in the new data, meant to increase the amount of "correct posture" data available, a weird spike was present in the right shoulder data. This could explain the superior predictive abilities of the right shoulder accelerometer. This spike in the data does not seem representative for the data as a whole, indicating a mistake in the measurement procedures. The "corrupted" data was removed from the dataset. In order to get a better idea of the usefulness of the different sensors, more data needs to be gathered. Both to replace the "corrupted" right shoulder data, but also to increase the amount of "correct posture" data.

6.2 Data Analysis session two

The goal of the second session was to expand upon the data of the first measuring session. This was done by gathering data from three more test subjects. These data gathering sessions focussed on obtaining extra correct posture data. This data was added to the data from the first data gathering session, resulting in one large dataset.

6.2.1 Training the random forest model using data relative to the correct posture.

In the same way as in the first measuring session, the data was processed by using a line of data from the test person sitting in a correct posture. This baseline data was subtracted from the other data, resulting in a delta between the correct baseline posture and the other postures. This data was then used to train and test the random forest model in Scikit learn. Eighty per cent of the data was used to train the model, while the remaining twenty per cent was used to test its accuracy. Just like in session number one, the confusion matrices from different models using different training and testing sets where added together in an attempt to create an accurate image of the accuracy of the model. See Figure 25 for the combined confusion matrix of the data from the 20 tree models.

	wrong	correct
wrong	215	1
correct	13	26

Figure 25, all confusion matrices of the 20 tree models added up, data relative to correct posture data. Green indicates the real class the data belongs to, yellow the predicted class.

Increasing the number of threes to 100 does not have a large effect on the accuracy of the model, see Figure 26.

	wrong	correct
wrong	216	0
correct	14	25

Figure 26, all confusion matrices of the 100 tree models added up, data relative to correct posture. Green indicates the real class the data belongs to, yellow the predicted class.

Again, the machine learning model has more difficulties predicting correct postures compared to incorrect postures. This could indicate a lack of correct posture data, even with the extra data gathered

during this measuring session, since only about twenty per cent of the data is from correct postures. Still, most of the correct postures are categorized correctly.

6.2.2 Analysing variable importance

Again, the build in variable importance measures of the random forest algorithm are used to get an idea of what sensors are the most important to posture measurement. See Figure 27 for the most important variables amongst various random test/train splits. The data in Figure 27 uses the 100 tree models to assure that all variables get enough use to give an accurate prediction of their importance.

Seed random split	0	1	2	3	4
Most important variable	24	24	24	24	24
Second most important variable	3	3	3	3	3
Third most important variable	5	5	5	5	5
Fourth most important variable	19	1	28	0	4
Fifth most important variable	10	28	0	4	27

Figure 27, top 5 most important variables for different train test splits, using 100 tree models.

Figure 27 clearly shows a high importance of variable 24, 3, and 5. The variable importance measures of the random forest models consistently place these variables at the top three of the variable importance lists. The fourth and fifth spot are more random with variable number 28, 0 and 4 being the most common.

6.2.2.1 Accuracy of variable 3, 5 and 24

variable 24, 3, and 5 correspond to the yaw of the upper back gyroscope, and the x and z axis of the left shoulder gyroscope respectively. Just like in the first data gathering session, one of the shoulder accelerometers is estimated to be of relatively high importance again. In order to test the usefulness of variable 3, 5 and 24, a machine learning model was trained using only these variables. The confusion matrix belonging to this model can be seen in Figure 28. Comparing this confusion matrix to the one in Figure 25, which uses all sensors, shows only a small sacrifice in accuracy when using only variable 24, 3 and 5.

	wrong	correct
wrong	212	4
correct	12	27

Figure 28, all confusion matrices of the 20 tree models added up, data relative to correct posture, using only variable 3, 5, and 24. Green indicates the real class the data belongs to, yellow the predicted class.

6.2.2.2 Accuracy of the model using only the left shoulder sensor

In an attempt to further reduce the number of sensors needed, a machine learning model was trained using only variables from the left shoulder sensor. First a model was trained using only the data from the left accelerometer, then a model was trained using both the accelerometer and the gyroscope of said shoulder, see figures 32 and 33. Again, a fairly small decrease in accuracy was detected.

	wrong	correct
wrong	209	7
correct	15	24

Figure 29, all confusion matrices of the 20 tree models added up, data relative to the correct posture, using only the accelerometer from the left shoulder. Green indicates the real class the data belongs to, yellow the predicted class.

	wrong	correct
wrong	211	5
correct	13	26

Figure 30, all confusion matrices of the 20 tree models added up, data relative to the correct posture, using only the gyroscope and accelerometer from the left shoulder. Green indicates the real class the data belongs to, yellow the predicted class

6.3 Data Analysis session three

The data from the third session is fully separate from the first two sessions. Measuring session three is designed to improve upon mistakes made in the first two sessions. A relatively larger amount of correct posture data was collected during measurements. Instead of taking a single line of data from each sensor, the data used during this session is an average of each sensor value over a 1 second window.

6.3.1 Training the random forest model using unprocessed data

First, the random forest model was trained using the 'raw' data. This data was not processed in any way. To test the influence of the number of threes in the random forest model on the accuracy of the model, both a 20-tree model and a 100-tree model where created using different random train/test splits. 20 per cent of the data was used to test the model while the other 80 per cent was used to train the model. Five different random train/test splits where used. The confusion matrices from these different splits where added together to create one combined confusion matrix, see Figure 34.

100-tree model: 86 per cent accurate

	wrong	correct
wrong	111	30
correct	12	157

20-tree model: 84 per cent accurate

	wrong	correct
wrong	110	31
correct	18	151

Figure 31, all confusion matrices of different models added up, using unprocessed data. Green indicates the real class the data belongs to, yellow the predicted class

The figure above shows a slight increase in accuracy when moving from a 20-tree model to a 100-tree model. The average accuracy of the 20-tree model is 0.84 compared to an average accuracy of 0.86 of the 100-tree models. This is to be expected based on the theory behind the random forest model, as more trees should result in a more accurate model. Because of the increased accuracy, any further tests will be done with 100 tree models.

interestingly, the overall accuracy of the machine learning model in the third session is lower than that of previous sessions. This can be explained by looking at the confusion matrices. Because of the additional correct posture data, there are more opportunities for the machine learning model to make mistakes. In the previous sessions, even a bad model could easily achieve 90 per cent accuracy by classifying everything as incorrect posture.

6.3.2 Training the random forest model using data relative to correct posture

As described in the specification chapter, it is expected that the accuracy of the machine learning model can be improved by relating the data to some sort of baseline. In the previous analysis this turned out to be the case. During this analysis however, the machine learning model already seems

quite accurate using the “raw” data. To test the usefulness of processing the data, the machine learning model was also trained using data relative to correct posture. Just like in the earlier test, the data was trained using five different train/test splits. 20 Per cent of the data was used to test the model while 80 per cent was used to train the model. The confusion matrices from these different splits where then added up to create one confusion matrix. Both a 20-tree model and a 100-tree model was trained, See Figure 32.

100 tree model: 92 per cent accurate

	wrong	correct
wrong	128	13
correct	11	143

20 tree model: 89 per cent accurate

	wrong	correct
wrong	124	17
correct	13	141

Figure 32, all confusion matrices of different models added up, using data relative to correct posture data. Green indicates the real class the data belongs to, yellow the predicted class

Both the 20-tree model and the 100-tree model show an increased accuracy over the models created using unprocessed data. Just like with the unprocessed data, the 100-tree model shows a slightly higher accuracy than the 20-tree model.

6.3.3 Training the random forest model using data relative to the upper back sensor.

Another way to process the sensor data is by relating it to one specific sensor. For example, it is possible to make all sensor positions relative to the sensor placed on the upper back of the test subject. This would result in a list of values containing the difference between the difference between the different sensors on the body and the upper back sensor. The benefit of this would be that it does not require a user to calibrate the device before use, something that would be necessary when using a device that uses correct posture data as its baseline. Another advantage would be that this setup could be less sensitive to the user moving around. A device that uses correct posture as a baseline is likely more sensitive to the user moving around and changing position at their desk or office.

When calculating these values, it is important to take into account that axis of the sensors might not line up correctly. For example, the x-axis of one sensor might be the Y-axis of another senor. In between each measurement it was assured that all the sensors remained in the same relative orientation to each other on the suit. Just like in the earlier data gathering sessions, using correct posture data as a baseline could help improve accuracy. The random forest model was trained using five different train/test splits, with 20 per cent of the data being reserved for testing each model. Each model used 100 trees. See Figure 33 for the results.

model with data relative to correct posture: 80 per cent accurate

	wrong	correct
wrong	96	40
correct	18	141

model with data not relative to correct posture: 76 per cent accurate

	wrong	correct
wrong	86	52
correct	24	154

Figure 33, all confusion matrices of different models added up, using data relative to the upper back sensor. 100 trees per model. Green indicates the real class the data belongs to, yellow the predicted class

Interestingly, the accuracy of both models in Figure 33 is smaller than the previous models. Even the models trained without the use of any kind of baseline manages to achieve a higher accuracy, see Figure 31. The inaccuracy of this model might be explained by the sensor not being perfectly lined up. Since the sensor suit was made out of stretchy fabric, and the sensors where attached using velcro, there might have been some deviations in the exact orientation of each sensor position. It is debatable whether these small errors in orientation would have such a significant effect on accuracy.

6.3.4 Analysing variable importance

The build in variable importance measures of the random forest algorithm will be used to estimate the importance of the different sensors on the sensor suit. During the analysis it was determined that the data relative to correct posture tends to produce the most accurate models. More specifically, the 100 tree models obtain the highest accuracy. For this reason, these models where used to estimate the variable importance. Figure 34 shows the five most important variables for five different random train/test splits.

Seed random split	0	1	2	3	4
Most important variable	14	14	14	14	1
Second most important variable	1	1	1	1	14
Third most important variable	8	2	2	2	8
Fourth most important variable	2	8	8	8	2
Fifth most important variable	7	7	32	7	7

Figure 34, top 5 most important variables for different train test splits, using 100 tree models.

Figure 34 shows that the variable importance of the most important variable does not vary much based on the train/test split. The most important variables, 14, 1, 2, 8, and 7, correspond to gyroscopes on the left and right shoulder and the head, see Figure 35.

variable	Sensor it belongs to
14	Head roll
1	Left shoulder yaw
2	Left shoulder pitch
8	Right shoulder pitch
7	Right shoulder yaw

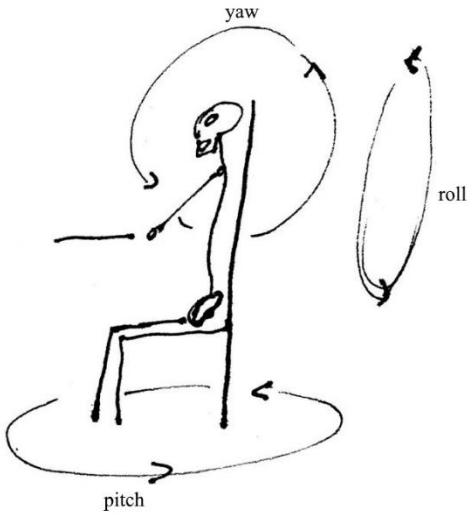


Figure 35, the most important variables and the gyroscopes they belong to.

6.3.5 Simplifying the machine learning model

Now that the most important sensors have been identified using variable importance estimations, it might be possible to simplify the machine learning model by removing the not important sensors.

6.3.4.1 Training a machine learning model using only the shoulder and head sensors

In part 6.3.3, the variable importance estimators estimated that the gyroscopes on the shoulders and head are the most important ones for creating an accurate machine learning model. To test the usefulness of a sensor suit that uses only sensor modules in these places, the random forest was trained using only the data from these sensor modules. The data used included both gyroscope and accelerometer data. All the data used correct posture data as a baseline. As with previous tests, multiple different train test splits where tested, their results where combined in the confusion matrix in Figure 36. 20 Per cent of the data was used to test each model, while the other 80 per cent was used to train said model. Each model used 100 trees.

	wrong	correct
wrong	120	16
correct	13	146

Figure 36, combined confusion matrices of 100 tree models using only data from both shoulders and the head sensor modules. data is relative to the correct sitting position. Green indicates the real class the data belongs to, yellow the predicted class.

The accuracy of the predictions in Figure 36 is 85 per cent. This is a bit lower than the model using all sensors (92 per cent). Of course, this setup still requires three sensor modules to set up. One might argue that having two sensor modules on opposite shoulders is unnecessary. To test the usefulness of having a sensor module on both shoulders, the machine learning model was trained using only the data from the left shoulder and the head, see Figure 37. The confusion matrix in Figure 37 is extremely similar to the one that uses the right shoulder data as well. The average accuracy of both the models that use the right shoulder data and the ones that do not is 85 per cent. This indicates that only one shoulder sensor is really needed. One thing that might screw the results of this test is that the different seating postures taken on during the test are symmetrical. This is a perfect situation for a single shoulder sensor. In real life it might happen that a user only shifts one shoulder in the wrong position, fooling the sensor or the other shoulder.

	wrong	correct
wrong	118	18
correct	11	148

Figure 37 ,combined confusion matrices of 100 tree models using only data from the left shoulder and head sensor modules. data is relative to the correct sitting position. Green indicates the real class the data belongs to, yellow the predicted class.

6.3.4.2 Testing the sensor modules individually.

To further simplify the model, the usefulness various sensor modules will be tested individually. Different models will be created using the data from the each of the shoulder modules and the head module. A model will be created based on the upper shoulder module as well. As it seems like this module might be able to combine the information from the shoulder and head modules due to its central positioning. The data from each sensor will be used to train 5 different models, each with a different train/test split. Each model will contain 100 trees. 20 Per cent of the data will be used to test the model, the other 80 per cent will be used to train the model. See Table 3 for the average accuracy obtained using the different sensors.

Sensor	**Average accuracy (out of 1.00)**
Left shoulder	0.85
Right shoulder	0.76
Head	0.74
Upper back	0.53

Table 3, average accuracy of the machine learning model created using each induvidual sensor

Even though the left and right shoulder can be used nearly interchangeably when combined with the head sensor, individually the left shoulder allows for a significantly more accurate model than the right one. The reason for this is not completely clear. Another notable result is the extremely low accuracy of the model created using only the upper back sensor. Based on its lower score in the variable importance test, this is not completely unexpected. It does however rule out the earlier hypothesis of the upper back sensor being able to combine all the best aspects of both the shoulder sensors and the head sensor.

6.3.4.3 Testing only the gyroscopes/accelerometers

To test the usefulness of using only the gyroscopes or only the accelerometers, different random forest models where trained using only the data from the gyroscopes or accelerometers. these models where created using the same parameters as the earlier models. This means they consist out of 100 trees and that 80 per cent of the data will be used to train each model. The other 20 per cent will be used to each the model. Like before. For both the gyroscopes and accelerometers, five different models using a different train/test splits where created. The data used during these tests uses the correct sitting position as a baseline. The confusion matrixes of all these models was added up to create a combined big confusion matrix, see Figure 38.

only accelerometers: 64 per cent accuracy

	wrong	correct
wrong	75	61
correct	45	99

only gyroscopes: 90 per cent accuracy

	wrong	correct
wrong	122	14
correct	15	144

Figure 38, all confusion matrices of different models added together. 100 trees per model. Green indicates the real class the data belongs to, yellow the predicted class.

The random forest models using only the gyroscope show a significantly higher accuracy than the models using only the accelerometers. With an average accuracy of 90 per cent, the models using only the gyroscopes are nearly as accurate as the models that use all data. The models created using only accelerometers however, achieve an accuracy that is barely higher than a random 50/50 guess.

6.2.4.4 Testing the data without the pitch angle

The data used to rain the random forest model assumes the user stays in the same basic position the whole time. In real lif3e, this is often not the case. For example, the average office chair is able to rotate around its pitch angle (see figure 38) to accommodate different seating orientations. When relating data to one line of correct posture data, this variation in the pitch angle is left out in favour of one “correct” pitch angle. One way to allow for the natural variation in pitch angle is by removing the relevant data altogether. For this test the pitch angles of all gyroscopes where removed from the dataset. The rest of the test remained the same as before. Figure 39 shows the combined confusion matrix based of the 5 different train/test splits that were used. The average accuracy of these models is 84 per cent. This is quite a bit lower than the 92 per cent accuracy that was obtained when including the pitch angles.

	wrong	correct
wrong	102	34
correct	14	145

Figure 39, all confusion matrices of different models added together, data set with pitch angle removed. 100 trees per model. Green indicates the real class the data belongs to, yellow the predicted class.

6.2.4.5 Testing only the stretch sensors

The previous tests focused on the data from the MPU 6050 sensor modules. However, the sensor suit also contained two stretch sensors. To test the usefulness of these sensors, a random forest model was

trained using only the data from the stretch sensors. This model was made using the same parameters as the previous models. 100 trees will be trained, using 80 percent of the data to train each random forest. The other twenty percent was used to test each model. 5 different models where trained using different train/test splits. The confusion matrices for each of these models where added up to create one big matrix, see

	wrong	correct
wrong	56	80
correct	65	94

Figure 40, all confusion matrices of different models added together. 100 trees per model, models trained using only the data from the stretch sensors. Green indicates the real class the data belongs to, yellow the predicted class.

The model trained using only the stretch sensors achieves an accuracy of no more than 51 per cent. Apparently, the predictive power of the model does not manage to eclipse that of a random 50/50 split.

6.4 Data analysis session four.

The fourth data gathering session differed in two ways from the third one. Firstly, the accelerometer data was stored in normalized spherical coordinates instead of Cartesian ones. This puts more focus on the direction of the acceleration while removing the magnitude of said acceleration. Just like during the third measuring session, the sensor values where averaged over a one second interval. In the case of the accelerator values, the average is taken using the normalized spherical coordinates, again removing the magnitude from the calculation.

The second way in which the fourth data gathering session differed from the last one is the way the measurements where conducted. To simulate the natural movements throughout the day that a person might make, the test subject is asked to rotate on an office chair in-between taking on the different postures.

6.4.1 Training the random forest model using unprocessed data.

The random forest model was trained using only the unprocessed data. This is the data straight from the sensor suit, without any processing done to it. A 20-tree and a 100-tree model was trained to compare their accuracies. The models were trained five different times, each time with a different train/test split. Each spit splits 80 per cent of the data into the training split, leaving the other 20 per cent for the testing of the model. The confusion matrices of the five different models where then added up to create an overview of the average accuracy of the model, see Figure 41.

100 tree model: 75 per cent accurate

	wrong	correct
wrong	99	45
correct	26	110

20 tree model: 75 per cent accurate

	wrong	correct
wrong	105	39
correct	32	104

Figure 41, all confusion matrices of different models added up, using unprocessed data. Green indicates the real class the data belongs to, yellow the predicted class.

The models in figure 41 show a lower accuracy to the models made using the unprocessed data from the third session. Presumable caused by the random rotation added to the measurements. The 100-tree model and the 20-tree model perform identically.

6.4.2 Training the random forest model using data relative to the correct posture.

In an attempt to make to random forest model more reliable, it was trained using data relative to correct posture data. Again a 20-tree and a 100-tree model was trained using different train/test splits. See Figure 42.

20 tree model: 84 per cent accurate

100 tree model: 84 per cent accurate

	wrong	correct
wrong	94	47
correct	10	214

	wrong	correct
wrong	96	45
correct	13	211

Figure 42, all confusion matrices of different models added up, using data relative to the correct posture. Green indicates the real class the data belongs to, yellow the predicted class

Figure 42 shows a large increase in accuracy when using the data relative to the correct posture compared to the unprocessed data. Just like with the unprocessed data, there is no difference in the accuracy of the 100-tree model and the 20-tree model. Interestingly, the accuracy of the model is the same as the model trained using the data from session 3 without the pitch angle (see part 6.2.4.4).

6.4.3 Analysing variable importance.

To get an idea of what sensors are the most useful, the random forest variable importance measures are used. The most important variables are determined using five different train/test splits. For each split a 100-tree model is created which is then used to determine the variable importance. Data that is relative to the correct posture was used, as it resulted in the most accurate models. See Figure 43.

Seed random split	0	1	2	3	4
Most important variable	11	26	1	1	1
Second most important variable	7	1	12	11	27
Third most important variable	26	2	2	7	2
Fourth most important variable	12	7	7	2	7
Fifth most important variable	1	12	26	27	11

Figure 43, top 5 most important variables for different train test splits, using 100 tree models.

Figure 43 shows a more divided spread of variable importance estimations than Figure 34 belonging to the previous data gathering session. Just like during session 3, all variables represented in the top 5 are from the gyroscopes. With variable 1 and 2 belonging to the left shoulder, variable 7 belonging to the right shoulder, variable 11 and 12 belonging to the headband, and variable 26 and 27 belonging to the lower back. Just like in the previous analysis, the shoulders and head sensors are very important. Interestingly, this time around the lower back sensor also gets a high rating.

6.4.3 Simplifying the machine learning model.

Just like in the previous analysis, the random forest model will be trained using only specific sensors in an attempt to decrease the complexity of the sensor suit.

6.4.3.1 training the model using only the shoulders, head, and lower back sensors.

The variable importance measures of the random forest model indicated that the gyroscopes of the shoulder, head and lower back are the most important ones for predicting posture quality. A random forest model was trained using only the data from the accelerometer and gyroscopes on these locations. The data used is relative to the correct posture. This model was trained five different times, using a different train/test split. The confusion matrices of these different models where combined to create Figure 44 Each model used 100 trees.

	wrong	correct
wrong	110	26
correct	21	123

Figure 44, combined confusion matrices of 100 tree models using only data from both shoulders, headband, and lower back sensors. data is relative to the correct sitting position. Green indicates the real class the data belongs to, yellow the predicted class.

Figure 44 shows an average accuracy of 83 per cent. This is virtually the same as the model that uses all sensors. It might be possible to decrease to complexity of the model further by removing one of the shoulder sensors, as the shoulder sensors are placed symmetrically. See Figure 45.

	wrong	correct
wrong	108	28
correct	24	120

Figure 45, combined confusion matrices of 100 tree models using only data from the left shoulder, headband, and lower back sensors. data is relative to the correct sitting position. Green indicates the real class the data belongs to, yellow the predicted class.

Figure 45 shows the confusion matrix from the model with the right shoulder removed. The accuracy of this model drops to 81 per cent. This is a bit lower than the model with the right shoulder included, but not by much.

The data analysis of the third measuring session concluded that a reasonable accurate model can be trained using only the left shoulder and the headband data. To test the accuracy of this setup using the new data, such a model was trained again. Again, the model uses data relative to the correct posture and multiple models using different train/tests splits where trained. See figure 46.

	wrong	correct
wrong	109	27
correct	25	119

Figure 46, combined confusion matrices of 100 tree models using only data from the left shoulder and headband sensor modules. Data is relative to the correct sitting position. Green indicates the real class the data belongs to, yellow the predicted class.

Figure 46 shows that removing the lower back sensor from the data does not have a negative effect on the predictive power of the model. In fact, the average accuracy of the predictions of the model stays the same as 81 per cent.

Instead of removing the lower back module from the dataset, on can also remove the shoulder module. To test the effectiveness of using the lower back sensor module instead of the shoulder one, a random forest model was trained with this data, using the same parameters as before. See figure 47.

	wrong	correct
wrong	101	35
correct	25	119

Figure 47, combined confusion matrices of 100 tree models using only data from the lower back and headband sensor modules. Data is relative to the correct sitting position. Green indicates the real class the data belongs to, yellow the predicted class.

The models created using only the data from the lower back and headband sensor modules have a lower average accuracy than the model using the left shoulder and headband modules, with 79 percent versus 81 per cent. However, the difference is quite small.

6.4.3.2 training the model using individual sensor modules

To simplify the model as much as possible, it was trained using only the data from one sensor module. The data that was used is relative to correct posture data, as this produces the most accurate

models. Only the left and right shoulder, head, and lower back modules were tested as these are the most important according to the variable importance estimations. The data from each sensor module was used to train 5 different models, each with a different train/test split. Each model contained 100 trees. 20 Per cent of the data was used to test the model, the other 80 per cent was used to train the model. See Table 4 for the average accuracy obtained using the different sensors.

Sensor	**Average accuracy (out of 1.00)**
Left shoulder	0.74
Right shoulder	0.72
Head	0.72
Upper back	0.75

Table 4, average accuracy of the machine learning model created using each individual sensor

Table 2 shows that the various sensors on their own can still achieve a reasonable accuracy in the low 70 per cent.

6.4.3.3 Testing the model using only the gyroscopes/accelerometers

The variable importance measures of the random forest model clearly favour the gyroscope data. In order to test the difference in usefulness between the gyroscopes and accelerometers, the machine learning model was trained using only the gyroscope and only the accelerometer data. Again, each model consisted out of 100 trees and the train/test split was 80/20. Processed data relative to the correct posture was used to train the models. Each model was trained five times using different train test splits and the results were added together in a confusion matrix, see Figure 48.

only accelerometers, 39 per cent accurate

	wrong	correct
wrong	47	89
correct	82	62

only gyroscopes, 84 per cent accurate

	wrong	correct
wrong	112	24
correct	22	122

Figure 48, all confusion matrices of different models added together. 100 trees per model. Green indicates the real class the data belongs to, yellow the predicted class.

Figure 48 shows an extremely low accuracy from the model using only gyroscopes of only 39 per cent. This is lower than you would expect from a random 50/50 guess and lower than the accelerometer only model trained using the data from the third measuring session. The extremely low accuracy of the gyroscope only model might be explained when looking at the changes made to the way the accelerometer data is recorded. Only the angles of the spherical coordinates are used instead of looking at the magnitude of the acceleration as well. This seems to have a detrimental effect on the usefulness of the gyroscope data.

The gyroscope model however, maintains the same accuracy of the model using both the gyroscope and the accelerometer. This confirms the suspicion that the accelerometer data is not useful for the machine learning model.

6.4.3.4 Testing only the stretch sensors.

In theory, the stretch sensor should have an advantage in the current measuring scenario since they should not be affected at al by the rotating of the test subject. However, the previous test has shown that training the model using only the gyroscopes, without the stretch sensors or accelerometers, has no effect on the accuracy of the model. To further test the usefulness of the stretch sensors a model was trained using only the stretch sensors.

	wrong	correct
wrong	46	90
correct	32	112

Figure 49, combined confusion matrices of 100 tree models using only data from the stretch sensors. Data is relative to the correct sitting position. Green indicates the real class the data belongs to, yellow the predicted class.

The model using only data from the stretch sensors achieves an accuracy of 56 per cent. This is just slightly higher than a random guess. Suggesting that the data from the stretch sensors does have a little bit of predictive ability, but not much.

7 Discussion.

7.1 Discussing the results.

The goal of this paper is to test the usefulness of machine learning, different sensors, and different sensor positions for posture identification. Various different data gathering sessions have been performed using a sensor suit that included stretch sensors, accelerometers, and gyroscopes. During these data gathering session various different postures where taken on by the test subjects. These postures where classified as good or bad and used to train a random forest machine learning algorithm.

In total four different data gathering sessions and one pilot session where conducted. The pilot session and first two data gathering sessions lead to various improvements of the measuring setup. The data from these sessions is not good enough to draw any meaningful conclusions. The data gathered during session three and four seems of high enough quality to draw useful conclusions.

Session three focussed on differentiating between different postures from a person sitting still at a desk, with no movement in between the different sitting positions. The fourth session tried to simulate someone moving throughout their office by having the test subject rotate around in between taking on the different postures. Session four also differs from the third one in the way it stored the accelerometer data. Instead of using Cartesian coordinates, the vectors from the accelerometer where converted to normalized spherical coordinates. This in an attempt to focus more on the direction of the acceleration than the intensity of it.

7.1.1 Effects of processing the data.

There are various ways to train the machine learning model. The data from the sensor suit can be used to train the machine learning model without any pre-processing being done to it. Because the gyroscopes reset every time they are turned on, there is no way of knowing in what position they really are relative to the earth or each other. On way to fix this is by using some kind of baseline. This baseline can consist of a line of correct posture data. Using this baseline data, the difference between the current posture and the correct posture can be calculated. When training the machine learning model using this data, a significant increase in accuracy of up to 10 percentage points is observed. The downside of such a model is that it required the user to sit up straight to calibrate the machine every time it is turned on. Another way to relate the data to some sort of baseline is by relating different sensor modules to each other. The difference in sensors values between different sensor modules could be useful for determining posture quality. The problem with this idea is that the starting position of the sensors is still not known, so some sort of calibration should still be needed to be able to relate the sensor values to each other. This suspicion was confirmed by the test results, as a model using data relative to the upper back sensor achieved an accuracy significantly lower than the model trained using data relative to the correct posture. It even scored lower than the model using the unprocessed data.

7.1.2 Simplifying the sensor suit.

One of the goals of this paper is to find an efficient sensor setup to measure posture quality. To do this the random forest variable importance measures where used. The algorithm was then trained and tested using only the data from the most promising sensors.

7.1.2.1 Usefulness of the different sensor types.

When analysing the data is became clear that the gyroscope data was by far the most useful out of the different sensor types. Training the model using only the gyroscope data resulted in a model nearly as accurate as the one using all data, this was true when using the data from either the third or the fourth measuring session. When training the data using only the accelerometer or stretch sensor data, the accuracy of the model decreased dramatically. The stretch sensors seem to be too limited in what they can measure, and too finicky to set up correctly, to be very useful when measuring posture. For better results using stretch sensors, it might work to add more of them to the sensor suit. However, this would also significantly increase the complexity of using the suit.

The model using only the accelerometer data from the fourth session achieved a significantly

lower accuracy compared the same model using the data from the third session. This can most likely be explained by the difference in which the accelerometer data was stored. Apparently, the magnitude of the acceleration vectors is important for the identification of the different postures. It is not known whether the magnitudes of said vectors actually contained useful information or whether random noise present in the accelerometer data from session three happened to match up with posture quality to a degree. Even the model from the accelerometer data from session 3 achieved only a 64 per cent accuracy.

In real life applications it is to be expected that the accelerometers would have even more trouble when identifying posture quality, as the used is likely to be busy moving around. For example, even typing on a keyboard could have significant effect on the values of the accelerometers on the shoulder.

7.1.2.2 Usefulness of the different sensor modules.

Using the variable importance measures from the random forest algorithm, the gyroscopes on the headband and both shoulder where consistently the most important. The models trained using the data from the fourth session also gave a high importance to the gyroscope on the lower back. When training the random forest algorithm using only the values from the shoulders and head band, a limited decrease in accuracy was observed. When training the machine learning model using the only the left shoulder data from the third session, no significant decrease in accuracy was observed versus using both shoulder and the headband. Doing the same but with the data from the fourth session did result in decreased accuracy. Interestingly, the data from only the lower back sensor in the fourth session was enough to train a reasonably accurate machine learning model, this model was just as accurate as the model trained using the left shoulder data. The lower back sensor data from the third session was not estimated to be as important.

7.2 Recommendation for further research.

This paper aims to give an insight in the usefulness of machine learning, different sensors, and different sensor positions for posture identification. The research done for this paper shows the usefulness of the random forest machine learning technique in combination with combination with gyroscopes placed on the shoulders and head. However, many interesting aspects of posture identification, machine learning, and wearable sensor technology have not been explored to their fullest potential.

The data gathered and analysed in this paper comes from a controlled measuring environment where the test subjects were asked to take on specific postures. In real life there is no clear difference between two different postures. The difference between a correct posture and one that is just slightly wrong are likely to be small and hard to detect. Users of a posture measuring device will likely shift and move around constantly. During the data analysis, even just adding random rotations to the measurements reduced the accuracy of the model by almost 10 percentage points. Detecting posture quality in these dynamic conditions is a challenge that requires further research.

The research done in this paper found relating the sensor data to correct posture data to be the most effective way to process the data as it resulted in the most accurate machine learning techniques. The downside of this technique is that it requires the user to calibrate the device at the start of each use by sitting correctly for a set amount of time. Developing a more streamlined way to calibrate the data would be beneficial to the development of a wearable posture sensor.

In this paper, only one type of machine learning algorithm was tested: the random forest algorithm. Random forests are known to be good performers and relatively easy to use, but they are not the only type of machine learning algorithm that can be useful for posture identification. Other algorithms that might be useful include, but are not limited to: K nearest neighbours, support vector machines, and gaussian mixture models. Further research could explore the effectiveness of different machine learning model for the task of posture identification.

Of course, the measuring setup used to gather the data for this paper is far from perfect. Even though it managed collect the data necessary, its practicality and durability leave something to be desired. If the aim is to develop a practical and useful wearable posture identification device, the sensor suit described in this report should not serve as a guide as it was designed as a quick and dirty solution to allow for the measuring sessions to take place.

7.3 Conclusion.

The aim of this report is to identify usefulness and efficiency of various wearable sensors and sensor positions in combination with machine learning techniques for measuring and identifying posture quality. In order to do this, correct and incorrect posture had to be defined. Correct posture was defined as posture that allows for efficient use of the muscles required to maintain said posture while avoiding unnecessary stress in the vertebrae and joints. Incorrect posture was defined as any kind of posture that is not correct posture. Based on these definitions various kinds of good and bad posture were identified.

Based on a review of the available literature, random forests was chosen to be a potentially useful machine learning algorithm. Random forests are known to be quite accurate and they are able to estimate variable importance. Measuring variable importance was useful for identifying the optimal sensors and sensor positions.

Various kinds of sensors and sensor positions were tested. Six gyroscope/accelerometer modules were placed on the pelvis, lower back, upper back, both shoulders, and head respectively. Two stretch sensors were placed, one on the upper back, following the shoulders, and one following the spine. Multiple measuring sessions were held to gather data on good and bad posture. Two of these measuring sessions resulted in a good, usable, dataset. The first one had all the test subjects face the same direction during the entire measuring session and stored the accelerometer data in spherical coordinates. The second session had the test subjects rotate randomly in between measurements to simulate a dynamic environment. This session also stored the accelerometer data in normalized spherical coordinates.

Based on the analysis of the data from the two test sessions, various things can be said about the usefulness of the sensors, sensor placement, and random forests. Firstly, the gyroscope data in both datasets tended to be far more useful for the machine learning algorithm than the accelerometer or stretch data. The accelerometer data in Cartesian coordinates was more useful than the data in the normalized spherical coordinates. Presumably, this means that for the gyroscope data the magnitude of the acceleration contains useful information. When using accelerometers in real life it is expected that their usefulness will go down even more due to an increase in random movements of the user.

The analysis of both datasets showed a high importance of the shoulder and head mounted gyroscopes. The second dataset, that included the random rotations in between measurements, showed in increased usefulness of the lower back sensor compared to the first. Using only one gyroscope on the left shoulder and one on the head resulted in a random forest model almost as accurate as the one using all sensors. For the dataset from the first test, without the random rotations, even using only the left shoulder gyroscope resulted in the same performance as using both the head mounted sensor and the left shoulder sensor. The shoulders seem to be one of the most useful spots on the body for mounting posture sensors.

In an attempt to increase the accuracy of the machine learning model, the data was pre-processed before training the machine learning model. Relating the values of the different gyroscopes and accelerometers to the gyroscopes and accelerometers on the upper back resulted in decreased accuracy over the model trained using untrained data. Relating the data from each sensor to data from the correct sitting position however, consistently showed an increase in accuracy of the machine learning model. This approach of processing the data helps solve the problem with the angle of the gyroscopes resetting every time they are turned on. The disadvantage of this approach is that it requires the user to calibrate the device every time it is turned on. It is to be expected that some way of calibrating the data will be even more useful in an real life use scenario due to decreased uniformity in

the way the measuring device will be treated by its users.

Overall, the data that included the random rotations resulted in decreased accuracy of the machine learning model. It is to be expected that a data set containing data from a real office environment will contain even more random noise, decreasing the accuracy of the model even further. Most likely, a larger amount of data will need to be gathered in order to create a model that is able to give consistently accurate predictions in a real working environment.

Random forests proved to be able to make fairly accurate predictions with regards to posture quality. The data analysis showed repeatedly that increasing the number of trees in the forest had a slightly positive impact on the accuracy of the predictions. Whether this increase in accuracy is worth the extra processing power will be depended on the specific use-case.

8 References

[1] L. Arundell, E. Fletcher, J. Salmon, J. Veitch, and T. Hinkley, "A systematic review of the prevalence of sedentary behavior during the after-school period among children aged 5-18 years," *Int. J. Behav. Nutr. Phys. Act.*, vol. 13, no. 1, p. 93, Dec. 2016.

[2] M. Neuhaus, G. N. Healy, D. W. Dunstan, N. Owen, and E. G. Eakin, "Workplace Sitting and Height-Adjustable Workstations: A Randomized Controlled Trial," *Am. J. Prev. Med.*, vol. 46, no. 1, pp. 30–40, Jan. 2014.

[3] H. Spittaels *et al.*, "Objectively measured sedentary time and physical activity time across the lifespan: a cross-sectional study in four age groups," *Int. J. Behav. Nutr. Phys. Act.*, vol. 9, no. 1, p. 149, Dec. 2012.

[4] C. Vandelanotte *et al.*, "Associations between occupational indicators and total, work-based and leisure-time sitting: a cross-sectional study," *BMC Public Health*, vol. 13, no. 1, p. 1110, Dec. 2013.

[5] C. E. Matthews *et al.*, "Amount of Time Spent in Sedentary Behaviors in the United States, 2003-2004," *Am. J. Epidemiol.*, vol. 167, no. 7, pp. 875–881, Mar. 2008.

[6] J. A. Bennie, J. Y. Chau, H. P. van der Ploeg, E. Stamatakis, A. Do, and A. Bauman, "The prevalence and correlates of sitting in European adults - a comparison of 32 Eurobarometer-participating countries," *Int. J. Behav. Nutr. Phys. Act.*, vol. 10, no. 1, p. 107, Sep. 2013.

[7] N. Gupta, C. S. Christiansen, D. M. Hallman, M. Korshøj, I. G. Carneiro, and A. Holtermann, "Is Objectively Measured Sitting Time Associated with Low Back Pain? A Cross-Sectional Investigation in the NOMAD study," *PLoS One*, vol. 10, no. 3, p. e0121159, Mar. 2015.

[8] P. B. O'Sullivan, T. Mitchell, P. Bulich, R. Waller, and J. Holte, "The relationship beween posture and back muscle endurance in industrial workers with flexion-related low back pain," *Man. Ther.*, vol. 11, no. 4, pp. 264–271, Nov. 2006.

[9] M. Noll, C. Tarragô Candotti, A. Vieira, and J. Fagundes Loss, "Back Pain and Body Posture Evaluation Instrument (BackPEI): development, content validation and reproducibility," *Int. J. Public Health*, vol. 58, no. 4, pp. 565–572, Aug. 2013.

[10] T. R. Oxland, "A history of spine biomechanics," *Unfallchirurg*, vol. 118, no. 1, pp. 80–92, 2015.

[11] F. Visser, "Haptic Feedback in a Posture Correcting Wearable," 2018.

[12] E. N. Corlett and J. A. E. Eklund, "How does a backrest work?," *Appl. Ergon.*, vol. 15, no. 2, pp. 111–114, 1984.

[13] J. Pelkey, "Upright posture and the meaning of meronymy: A synthesis of metaphoric and analytic accounts," *Cogn. Semiot.*, vol. 11, no. 1, pp. 1–22, 2018.

[14] "THE EXAMINATION OF THE SPINE," *Mediview*. [Online]. Available: https://simbrazil.mediviewprojects.org/index.php/spinal-shape-and-posture-defects/sagittal-plane-deformities. [Accessed: 31-Mar-2019].

[15] K. Hajizadeh, I. Gibson, and G. Liu, "Developing a 3D multi-body model of the scoliotic spine with lateral bending motion for comparison of ribcage flexibility," *Int. J. Adv. Des. Manuf. Technol.*, vol. 6, no. 1, pp. 25–32, 2012.

[16] M. A. Adams, A. F. Mannion, and P. Dolan, "Personal Risk Factors for First-Time Low Back Pain," vol. 24, no. 23, pp. 2497–2505, 1999.

[17] Y. Brink, Q. Louw, K. Grimmer, and E. Jordaan, "The relationship between sitting posture and seated-related upper quadrant musculoskeletal pain in computing South African adolescents: A

prospective study," *Man. Ther.*, vol. 20, no. 6, pp. 820–826, Dec. 2015.

[18] L. Straker, R. Skoss, A. Burnett, and R. Burgess-Limerick, "Effect of visual display height on modelled upper and lower cervical gravitational moment, muscle capacity and relative strain," *Ergonomics*, vol. 52, no. 2, pp. 204–221, Feb. 2009.

[19] M. Thuresson, B. Äng, J. Linder, and K. Harms-Ringdahl, "Mechanical load and EMG activity in the neck induced by different head-worn equipment and neck postures," *Int. J. Ind. Ergon.*, vol. 35, no. 1, pp. 13–18, Jan. 2005.

[20] S. J. Edmondston, M. Sharp, A. Symes, N. Alhabib, and G. T. Allison, "Changes in mechanical load and extensor muscle activity in the cervico-thoracic spine induced by sitting posture modification," *Ergonomics*, vol. 54, no. 2, pp. 179–186, Feb. 2011.

[21] F. Attal, S. Mohammed, M. Dedabrishvili, F. Chamroukhi, L. Oukhellou, and Y. Amirat, "Physical human activity recognition using wearable sensors," *Sensors (Switzerland)*, vol. 15, no. 12, pp. 31314–31338, 2015.

[22] J. Pansiot, D. Stoyanov, D. Mcilwraith, B. P. L. Lo, and G. Z. Yang, "Ambient and Wearable Sensor Fusion for Activity Recognition in Healthcare Monitoring Systems."

[23] S. Patel, C. Mancinelli, J. Healey, M. Moy, and P. Bonato, "Using Wearable Sensors to Monitor Physical Activities of Patients with COPD: A Comparison of Classifier Performance," in *2009 Sixth International Workshop on Wearable and Implantable Body Sensor Networks*, 2009, pp. 234–239.

[24] D. Rodriguez-Martin, A. Samà, C. Perez-Lopez, A. Català, J. Cabestany, and A. Rodriguez-Molinero, "SVM-based posture identification with a single waist-located triaxial accelerometer," *Expert Syst. Appl.*, vol. 40, no. 18, pp. 7203–7211, 2013.

[25] K. J. Archer and R. V Kimes, "Empirical characterization of random forest variable importance measures," *Comput. Stat. Data Anal.*, vol. 52, pp. 2249–2260, 2008.

[26] Leo Breiman, "Manual on setting up, using, and understanding random forests V3.1."

[27] A. Liaw and M. Wiener, "Classification and Regression by randomForest," vol. 2, no. 5, pp. 18–22, 2002.

[28] M. Belgiu and L. Drăgut, "Random forest in remote sensing: A review of applications and future directions," *ISPRS J. Photogramm. Remote Sens.*, vol. 114, pp. 24–31, 2016.

[29] C. Strobl, A. Boulesteix, A. Zeileis, and T. Hothorn, "Bias in random forest variable importance measures: Illustrations, sources and a solution," *BMC Bioinforma. 2007*, vol. 8, no. 25, 2007.

[30] "R-project: www.r-project.org." .

[31] É. D. Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, "Scikit-learn: Machine Learning in Python." Journal of Machine Learning Research, p. 2825−2830, 2011.

[32] L. E. O. Breiman, "Random Forest," *Mach. Learn.*, pp. 5–32, 2001.

[33] K. I. Usama Fayyad, "The attribute selection problem in decision tree generation," *AAAI*, 1992.

[34] L. Breiman, "Bagging predictors," *Mach. Learn.*, vol. 140, pp. 123–140, 1996.

[35] L. Breiman, "Out-of-Bag Estimation," *Berkeley Univ. Calif. Stat. Dep.*, 1996.

[36] "Lumo Lift – Lumo Bodytech." [Online]. Available: https://support.lumobodytech.com/hc/en-

us/categories/201535363-Lumo-Lift. [Accessed: 17-Apr-2019].

[37] “How To Improve Posture With Upright - Fix Bad Posture Fast!” [Online]. Available: https://www.uprightpose.com/how-it-works/. [Accessed: 17-Apr-2019].

[38] “Opter - Augmented Vitality.” [Online]. Available: https://opterlife.com/. [Accessed: 17-Apr-2019].

[39] “Darma.co.” [Online]. Available: http://darma.co/index.html. [Accessed: 17-Apr-2019].

[40] “Xsens MVN Analyze - Products - Xsens 3D motion tracking.” [Online]. Available: https://www.xsens.com/products/xsens-mvn-analyze/. [Accessed: 17-Apr-2019].

[41] B. V. Xsens Technologies, “Xsens MVN User Manual,” *User Guid. Xsens MVN, MVN Link, MVN Awinda*, no. October, 2018.

[42] “MPU-6000 and MPU-6050 Product Specification Revision 3.1.”

[43] “Adafruit TCA9548A 1-to-8 I2C Multiplexer Breakout | Adafruit Learning System.” [Online]. Available: https://learn.adafruit.com/adafruit-tca9548a-1-to-8-i2c-multiplexer-breakout/overview. [Accessed: 23-May-2019].

[44] “Conductive Rubber Cord Stretch Sensor + extras! ID: 519 - $9.95 : Adafruit Industries, Unique & fun DIY electronics and kits.” [Online]. Available: https://www.adafruit.com/product/519. [Accessed: 23-May-2019].

[45] “Arduino Nano.” [Online]. Available: https://store.arduino.cc/arduino-nano. [Accessed: 23-May-2019].

[46] J. Roberg, “i2cdevlib @ github.com.” .

[47] “Wiring & Test | Adafruit TCA9548A 1-to-8 I2C Multiplexer Breakout | Adafruit Learning System.” [Online]. Available: https://learn.adafruit.com/adafruit-tca9548a-1-to-8-i2c-multiplexer-breakout/wiring-and-test. [Accessed: 12-Jun-2019].

[48] “Random Forest Algorithm with Python and Scikit-Learn.” [Online]. Available: https://stackabuse.com/random-forest-algorithm-with-python-and-scikit-learn/. [Accessed: 12-Jun-2019].

9 Appendix

Appendix A, python machine learning code.

```
# -*- coding: utf-8 -*-
"""
Created on Wed May 15 14:57:33 2019

@author: User
"""

import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier as RFC
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

import matplotlib.pyplot as plt

#dataset has 39 columns
dataset = pd.read_excel('FILENAME.xlsx')
#print(dataset)

#x = attribute, y = lable
X = dataset.iloc[:, 1:39].values
y = dataset.iloc[:, 0].values

#split the data in testing and training sets.
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_stat
    e=4)
#print(X_train)
#scale the data
#sc = StandardScaler()
#X_train = sc.fit_transform(X_train)
#X_test = sc.transform(X_test)

#train the algorithm
forest = RFC(n_estimators=100, random_state=0)
forest.fit(X_train, y_train)
y_pred = forest.predict(X_test)


#for input, prediction, label in zip(X_test, y_pred, y_test):
#  if prediction != label:
#    print(input, 'has been classified as ', prediction, 'and should be ', label)

#test accuracy
print(confusion_matrix(y_test,y_pred))
print(classification_report(y_test,y_pred))
print(accuracy_score(y_test, y_pred))

importances = forest.feature_importances_

std = np.std([tree.feature_importances_ for tree in forest.estimators_],
             axis=0)
indices = np.argsort(importances)[::-1]

#Print the feature ranking
print("Feature ranking:")

for f in range(X.shape[1]):
    print("%d. feature %d (%f)" % (f + 1, indices[f], importances[indices[f]]))

#Plot the feature importances of the forest
```

```
60. plt.figure()
61. plt.title("Feature importances")
62. plt.bar(range(X.shape[1]), importances[indices],
63.        color="r", yerr=std[indices], align="center")
64. plt.xticks(range(X.shape[1]), indices)
65. plt.xlim([-1, X.shape[1]])
66. plt.show()
```

Appendix B, Arduino code sensor suit.

Appendix B-I, first version of the Arduino code.

This version of the coda does not use a moving window. The accelerometer data is stored in Cartesian coordinates.

```
// code made by Gijs van Rhijn, 2019
// This code is based on code by Jeff Roberg
/* ============================================
  I2Cdev device library code is placed under the MIT license
  Copyright (c) 2012 Jeff Rowberg

  Permission is hereby granted, free of charge, to any person obtaining a copy
  of this software and associated documentation files (the "Software"), to deal
  in the Software without restriction, including without limitation the rights
  to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
  copies of the Software, and to permit persons to whom the Software is
  furnished to do so, subject to the following conditions:

  The above copyright notice and this permission notice shall be included in
  all copies or substantial portions of the Software.

  THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
  IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
  FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
  AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
  LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
  OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
  THE SOFTWARE.
  ===============================================
*/
// The code for the Adafruit TCA9548A multiplexer is based on a tutorial by Adafruit : https://learn.adafruit.com/adafruit-tca9548a-1-to-8-i2c-multiplexer-breakout/wiring-and-test

// I2Cdev and MPU6050 must be installed as libraries, or else the .cpp/.h files
// for both classes must be in the include path of your project
#include "I2Cdev.h"

#include "MPU6050_6Axis_MotionApps20.h"
//#include "MPU6050.h" // not necessary if using MotionApps include file

// Arduino Wire library is required if I2Cdev I2CDEV_ARDUINO_WIRE implementation
// is used in I2Cdev.h
#if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
#include "Wire.h"
#endif

// class default I2C address is 0x68
// specific I2C addresses may be passed as a parameter here
// AD0 low = 0x68 (default for SparkFun breakout and InvenSense evaluation board)
// AD0 high = 0x69
MPU6050 mpu;
MPU6050 mpu2;
//MPU6050 mpu(0x68); // <-- use for AD0 high

/* =========================================================================
   NOTE: In addition to connection 3.3v, GND, SDA, and SCL, this sketch
   depends on the MPU-6050's INT pin being connected to the Arduino's
   external interrupt #0 pin. On the Arduino Uno and Mega 2560, this is
   digital I/O pin 2.
   ========================================================================= */

/* =========================================================================
   NOTE: Arduino v1.0.1 with the Leonardo board generates a compile error
```

```
58.    when using Serial.write(buf, len). The Teapot output uses this method.
59.    The solution requires a modification to the Arduino USBAPI.h file, which
60.    is fortunately simple, but annoying. This will be fixed in the next IDE
61.    release. For more info, see these links:
62.
63.    http://arduino.cc/forum/index.php/topic,109987.0.html
64.    http://code.google.com/p/arduino/issues/detail?id=958
65.    ========================================================================= */
66.
67.
68.
69. // uncomment "OUTPUT_READABLE_QUATERNION" if you want to see the actual
70. // quaternion components in a [w, x, y, z] format (not best for parsing
71. // on a remote host such as Processing or something though)
72. //#define OUTPUT_READABLE_QUATERNION
73.
74. // uncomment "OUTPUT_READABLE_EULER" if you want to see Euler angles
75. // (in degrees) calculated from the quaternions coming from the FIFO.
76. // Note that Euler angles suffer from gimbal lock (for more info, see
77. // http://en.wikipedia.org/wiki/Gimbal_lock)
78. //#define OUTPUT_READABLE_EULER
79.
80. // uncomment "OUTPUT_READABLE_YAWPITCHROLL" if you want to see the yaw/
81. // pitch/roll angles (in degrees) calculated from the quaternions coming
82. // from the FIFO. Note this also requires gravity vector calculations.
83. // Also note that yaw/pitch/roll angles suffer from gimbal lock (for
84. // more info, see: http://en.wikipedia.org/wiki/Gimbal_lock)
85. //#define OUTPUT_READABLE_YAWPITCHROLL //this one for gyro
86.
87. // uncomment "OUTPUT_READABLE_REALACCEL" if you want to see acceleration
88. // components with gravity removed. This acceleration reference frame is
89. // not compensated for orientation, so +X is always +X according to the
90. // sensor, just without the effects of gravity. If you want acceleration
91. // compensated for orientation, us OUTPUT_READABLE_WORLDACCEL instead.
92. //#define OUTPUT_READABLE_REALACCEL
93.
94. // uncomment "OUTPUT_READABLE_WORLDACCEL" if you want to see acceleration
95. // components with gravity removed and adjusted for the world frame of
96. // reference (yaw is relative to initial orientation, since no magnetometer
97. // is present in this case). Could be quite handy in some cases.
98. //#define OUTPUT_READABLE_WORLDACCEL // this one for acccel
99.
100.       // uncomment "OUTPUT_TEAPOT" if you want output that matches the
101.       // format used for the InvenSense teapot demo
102.       //#define OUTPUT_TEAPOT
103.
104.
105.
106.       #define LED_PIN 13 // (Arduino is 13, Teensy is 11, Teensy++ is 6)
107.       bool blinkState = false;
108.
109.       // MPU control/status vars
110.       bool dmpReady = false;  // set true if DMP init was successful
111.       uint8_t mpuIntStatus;   // holds actual interrupt status byte from MPU
112.       uint8_t devStatus;      // return status after each device operation (0 = suc
    cess, !0 = error)
113.       uint16_t packetSize;    // expected DMP packet size (default is 42 bytes)
114.       uint16_t fifoCount;     // count of all bytes currently in FIFO
115.       uint8_t fifoBuffer[64]; // FIFO storage buffer
116.
117.       // orientation/motion vars
118.       Quaternion q;           // [w, x, y, z]         quaternion container
119.       VectorInt16 aa;         // [x, y, z]            accel sensor measurements
120.       VectorInt16 aaReal;     // [x, y, z]            gravity-
    free accel sensor measurements
```

```
121.       VectorInt16 aaWorld;    // [x, y, z]            world-
    frame accel sensor measurements
122.       VectorFloat gravity;    // [x, y, z]            gravity vector
123.       float euler[3];         // [psi, theta, phi]    Euler angle container
124.       float ypr[3];           // [yaw, pitch, roll]   yaw/pitch/roll container and
    gravity vector
125.
126.       // packet structure for InvenSense teapot demo
127.       uint8_t teapotPacket[14] = { '$', 0x02, 0, 0, 0, 0, 0, 0, 0, 0, 0x00, 0x00, '
    \r', '\n' };
128.
129.       // ================================================================
130.       // ===               VARIABLES NEEDED FOR MULTIPLE MPUs          ===
131.       // ================================================================
132.       //#include "Arduino.h"
133.       //#include "class-MpuData.h";
134.       #include <MpuData.h>
135.       const byte gyroCount = 6; //amount of gyros connected
136.       byte gyro = 0; //current gyro being read by the arduino
137.       int mpuData[gyroCount][6] = {0}; //2 dimensional array containing the data fo
    r each gyro (yaw, pitch roll, x, y, z)
138.
139.       // ================================================================
140.       // ===               VARIABLES NEEDED FOR SERIAL COM             ===
141.       // ================================================================
142.
143.       bool snd = false;
144.
145.
146.       // ================================================================
147.       // ===                INTERRUPT DETECTION ROUTINE                ===
148.       // ================================================================
149.
150.       volatile bool mpuInterrupt = false;     // indicates whether MPU interrupt pi
    n has gone high
151.       void dmpDataReady() {
152.         mpuInterrupt = true;
153.       }
154.
155.
156.
157.       // ================================================================
158.       // ===                      INITIAL SETUP                       ===
159.       // ================================================================
160.
161.       void setup() {
162.         // join I2C bus (I2Cdev library doesn't do this automatically)
163.       #if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
164.         Wire.begin();
165.         TWBR = 24; // 400kHz I2C clock (200kHz if CPU is 8MHz)
166.       #elif I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_FASTWIRE
167.         Fastwire::setup(400, true);
168.       #endif
169.         tcaselect(0); // select the right gyro
170.
171.         // initialize serial communication
172.         // (115200 chosen because it is required for Teapot Demo output, but it's
173.         // really up to you depending on your project)
174.         Serial.begin(115200);
175.         while (!Serial); // wait for Leonardo enumeration, others continue immediat
    ely
176.
177.         // NOTE: 8MHz or slower host processors, like the Teensy @ 3.3v or Ardunio

178.         // Pro Mini running at 3.3v, cannot handle this baud rate reliably due to
```

```
        // the baud timing being too misaligned with processor ticks. You must use
        // 38400 or slower in these cases, or use some kind of external separate
        // crystal solution for the UART timer.

        // initialize device in loop
        Serial.println(F("Initializing I2C devices..."));
        for (int i = 0; i < gyroCount; i++) {
          tcaselect(i); // select the right gyro
          mpu.initialize();
          Serial.print("initializing: ");
          Serial.println(i);
        }
        tcaselect(0); // select the right gyro

        // verify connection in loop
        Serial.println(F("Testing device connections..."));
        for (int i = 0; i < gyroCount; i++) {
          tcaselect(i); // select the right gyro
          Serial.println(mpu.testConnection() ? F("MPU6050 connection successful") : F("MPU6050 connection failed"));
        }
        tcaselect(0); // select the right gyro

        // wait for ready
        Serial.println(F("\nSend any character to begin DMP programming and demo: "));
        while (Serial.available() && Serial.read()); // empty buffer
        while (!Serial.available());                 // wait for data
        while (Serial.available() && Serial.read()); // empty buffer again

        // load and configure the DMP in loop
        Serial.println(F("Initializing DMP..."));
        for (int i = 0; i < gyroCount; i++) {
          tcaselect(i); // select the right gyro
          devStatus = mpu.dmpInitialize();
        }
        tcaselect(0); // select the right gyro

        // supply your own gyro offsets here, scaled for min sensitivity
        mpu.setXGyroOffset(220);
        mpu.setYGyroOffset(76);
        mpu.setZGyroOffset(-85);
        mpu.setZAccelOffset(1788); // 1688 factory default for my test chip


        // make sure it worked (returns 0 if so)
        if (devStatus == 0) {
          // turn on the DMP, now that it's ready
          Serial.println(F("Enabling DMP..."));
          for (int i = 0; i < gyroCount; i++) {
            tcaselect(i); // select the right gyro
            mpu.setDMPEnabled(true);
          }
          tcaselect(0); // select the right gyro

          // enable Arduino interrupt detection
          Serial.println(F("Enabling interrupt detection (Arduino external interrupt 0)..."));
          attachInterrupt(0, dmpDataReady, RISING);
          mpuIntStatus = mpu.getIntStatus();

          // set our DMP Ready flag so the main loop() function knows it's okay to use it
          Serial.println(F("DMP ready! Waiting for first interrupt..."));
          dmpReady = true;
```

```
240.          //
241.          // get expected DMP packet size for later comparison
242.          packetSize = mpu.dmpGetFIFOPacketSize();
243.        } else {
244.          // ERROR!
245.          // 1 = initial memory load failed
246.          // 2 = DMP configuration updates failed
247.          // (if it's going to break, usually the code will be 1)
248.          Serial.print(F("DMP Initialization failed (code "));
249.          Serial.print(devStatus);
250.          Serial.println(F(")"));
251.        }
252.
253.        // configure LED for output
254.        pinMode(LED_PIN, OUTPUT);
255.
256.      }
257.
258.      void tcaselect(uint8_t i) {
259.        if (i > 7) return;
260.
261.        Wire.beginTransmission(0x70);
262.        Wire.write(1 << i);
263.        Wire.endTransmission();
264.      }
265.
266.      // ================================================================
267.      // ===                      MAIN PROGRAM LOOP                   ===
268.      // ================================================================
269.
270.      void loop() {
271.        gyro = gyro % gyroCount;
272.        checkSnd(); //check if there is a message from the serial port
273.
274.        //  if (gyro == 0) {
275.        //    gyro = 1;
276.        //  }
277.        //  else gyro = 0;
278.
279.        tcaselect(gyro); // select the right gyro
280.
281.        // if programming failed, don't try to do anything
282.        if (!dmpReady) return;
283.
284.        // wait for MPU interrupt or extra packet(s) available
285.        while (!mpuInterrupt && fifoCount < packetSize) {
286.          // other program behavior stuff here
287.          // .
288.          // .
289.          // .
290.          // if you are really paranoid you can frequently test in between other
291.          // stuff to see if mpuInterrupt is true, and if so, "break;" from the
292.          // while() loop to immediately process the MPU data
293.          // .
294.          // .
295.          // .
296.        }
297.
298.        // reset interrupt flag and get INT_STATUS byte
299.        mpuInterrupt = false;
300.        mpuIntStatus = mpu.getIntStatus();
301.        // get current FIFO count
302.        fifoCount = mpu.getFIFOCount();
303.        //  Serial.println(fifoCount);
304.
```

```
305.         // check for overflow (this should never happen unless our code is too inef
    ficient)
306.         if ((mpuIntStatus & 0x10) || fifoCount == 1024) {
307.           // reset so we can continue cleanly
308.           mpu.resetFIFO();
309.           //    Serial.println(F("FIFO overflow!"));
310.
311.           // otherwise, check for DMP data ready interrupt (this should happen freq
    uently)
312.         } else if (mpuIntStatus & 0x02) {
313.           // wait for correct available data length, should be a VERY short wait
314.           while (fifoCount < packetSize) fifoCount = mpu.getFIFOCount();
315.
316.           // read a packet from FIFO
317.           mpu.getFIFOBytes(fifoBuffer, packetSize);
318.
319.           // track FIFO count here in case there is > 1 packet available
320.           // (this lets us immediately read more without waiting for an interrupt)

321.           fifoCount -= packetSize;
322.
323.           //    Serial.print("gyro: ");
324.           //    Serial.println(gyro);
325.
326.           saveData();
327.
328.           if (snd) {
329.             snd = false;
330.             //      saveData();
331.             //      Serial.print("YPR brah: ");
332.             //      Serial.println(mpuData[0][0]);
333.             //      Serial.println(mpuData[0][1]);
334.             //      Serial.println(mpuData[0][2]);
335.             //      Serial.println(mpuData[0][3]);
336.             //      Serial.println(mpuData[0][4]);
337.             //      Serial.println(mpuData[0][5]);
338.             for (int g = 0; g < gyroCount; g++) {
339.       //        Serial.print("G");
340.       //        Serial.print(g);
341.       //        Serial.print(";");
342.       //        Serial.print(mpuData[g][0]);
343.       //        Serial.print(";");
344.       //        Serial.print(mpuData[g][1]);
345.       //        Serial.print(";");
346.       //        Serial.print(mpuData[g][2]);
347.       //        Serial.print(";");
348.               Serial.print(mpuData[g][3]);
349.               Serial.print(";");
350.       //        Serial.print(mpuData[g][4]);
351.       //        Serial.print(";");
352.       //        Serial.print(mpuData[g][5]);
353.       //        Serial.print(";");
354.
355.               //        break;
356.             }
357.       //      Serial.print(analogRead(A3));
358.       //      Serial.print(";");
359.       //      Serial.print(analogRead(A6));
360.             Serial.println("");
361.
362.       //      Serial.print(analogRead(";"));
363.       //      Serial.print(analogRead(A6));
364.       //      Serial.println("");
365.
366.             //      Serial.print();
367.             //      Serial.print();
```

```
368.        #ifdef OUTPUT_READABLE_QUATERNION
369.              // display quaternion values in easy matrix form: w x y z
370.              mpu.dmpGetQuaternion(&q, fifoBuffer);
371.              Serial.print("quat\t");
372.              Serial.print(q.w);
373.              Serial.print("\t");
374.              Serial.print(q.x);
375.              Serial.print("\t");
376.              Serial.print(q.y);
377.              Serial.print("\t");
378.              Serial.println(q.z);
379.        #endif
380.
381.        #ifdef OUTPUT_READABLE_EULER
382.              // display Euler angles in degrees
383.              mpu.dmpGetQuaternion(&q, fifoBuffer);
384.              mpu.dmpGetEuler(euler, &q);
385.              Serial.print("euler\t");
386.              Serial.print(euler[0] * 180 / M_PI);
387.              Serial.print("\t");
388.              Serial.print(euler[1] * 180 / M_PI);
389.              Serial.print("\t");
390.              Serial.println(euler[2] * 180 / M_PI);
391.        #endif
392.
393.        #ifdef OUTPUT_READABLE_YAWPITCHROLL
394.              // display Euler angles in degrees
395.              mpu.dmpGetQuaternion(&q, fifoBuffer);
396.              mpu.dmpGetGravity(&gravity, &q);
397.              mpu.dmpGetYawPitchRoll(ypr, &q, &gravity);
398.              Serial.print("gyro");
399.              Serial.print(gyro);
400.              Serial.print("\t");
401.              Serial.print(ypr[0] * 180 / M_PI);
402.              Serial.print("\t");
403.              Serial.print(ypr[1] * 180 / M_PI);
404.              Serial.print("\t");
405.              Serial.println(ypr[2] * 180 / M_PI);
406.        #endif
407.
408.        #ifdef OUTPUT_READABLE_REALACCEL
409.              // display real acceleration, adjusted to remove gravity
410.              mpu.dmpGetQuaternion(&q, fifoBuffer);
411.              mpu.dmpGetAccel(&aa, fifoBuffer);
412.              mpu.dmpGetGravity(&gravity, &q);
413.              mpu.dmpGetLinearAccel(&aaReal, &aa, &gravity);
414.              Serial.print("areal\t");
415.              Serial.print(aaReal.x);
416.              Serial.print("\t");
417.              Serial.print(aaReal.y);
418.              Serial.print("\t");
419.              Serial.println(aaReal.z);
420.        #endif
421.
422.        #ifdef OUTPUT_READABLE_WORLDACCEL
423.              // display initial world-
    frame acceleration, adjusted to remove gravity
424.              // and rotated based on known orientation from quaternion
425.              mpu.dmpGetQuaternion(&q, fifoBuffer);
426.              mpu.dmpGetAccel(&aa, fifoBuffer);
427.              mpu.dmpGetGravity(&gravity, &q);
428.              mpu.dmpGetLinearAccel(&aaReal, &aa, &gravity);
429.              mpu.dmpGetLinearAccelInWorld(&aaWorld, &aaReal, &q);
430.              Serial.print("aworld\t");
431.              Serial.print(aaWorld.x);
432.              Serial.print("\t");
```

```
433.            Serial.print(aaWorld.y);
434.            Serial.print("\t");
435.            Serial.println(aaWorld.z);
436.        #endif
437.
438.        #ifdef OUTPUT_TEAPOT
439.            // display quaternion values in InvenSense Teapot demo format:
440.            teapotPacket[2] = fifoBuffer[0];
441.            teapotPacket[3] = fifoBuffer[1];
442.            teapotPacket[4] = fifoBuffer[4];
443.            teapotPacket[5] = fifoBuffer[5];
444.            teapotPacket[6] = fifoBuffer[8];
445.            teapotPacket[7] = fifoBuffer[9];
446.            teapotPacket[8] = fifoBuffer[12];
447.            teapotPacket[9] = fifoBuffer[13];
448.            Serial.write(teapotPacket, 14);
449.            teapotPacket[11]++; // packetCount, loops at 0xFF on purpose
450.        #endif
451.            // blink LED to indicate activity
452.            blinkState = !blinkState;
453.            digitalWrite(LED_PIN, blinkState);
454.          }
455.        }
456.        gyro++;
457.      }
458.
459.      void checkSnd() {
460.        if (Serial.available()) {
461.          Serial.read();
462.          snd = true;
463.        }
464.      }
465.
466.      //void sendMsg() {
467.      //  //implement
468.      //}
469.
470.      void saveData() {
471.        // display Euler angles in degrees
472.        mpu.dmpGetQuaternion(&q, fifoBuffer);
473.        mpu.dmpGetGravity(&gravity, &q);
474.        mpu.dmpGetYawPitchRoll(ypr, &q, &gravity);
475.
476.        int yaw = ypr[0] * 180 / M_PI;
477.        int pitch = ypr[1] * 180 / M_PI;
478.        int roll = ypr[2] * 180 / M_PI;
479.        mpuData [gyro] [0] = yaw;
480.        mpuData [gyro] [1] = pitch;
481.        mpuData [gyro] [2] = roll;
482.
483.        // display initial world-frame acceleration, adjusted to remove gravity
484.        // and rotated based on known orientation from quaternion
485.        mpu.dmpGetQuaternion(&q, fifoBuffer);
486.        mpu.dmpGetAccel(&aa, fifoBuffer);
487.        mpu.dmpGetGravity(&gravity, &q);
488.        mpu.dmpGetLinearAccel(&aaReal, &aa, &gravity);
489.        mpu.dmpGetLinearAccelInWorld(&aaWorld, &aaReal, &q);
490.        //  Serial.print("aworld\t");
491.        //  Serial.print(aaWorld.x);
492.        //  Serial.print("\t");
493.        //  Serial.print(aaWorld.y);
494.        //  Serial.print("\t");
495.        //  Serial.println(aaWorld.z);
496.
497.        int x = aaWorld.x;
498.        int y = aaWorld.y;
```

```
        int z = aaWorld.z;
        mpuData [gyro] [3] = x;
        mpuData [gyro] [4] = y;
        mpuData [gyro] [5] = z;
      }
```

Appendix B-II, second version of the Arduino code.

This version of the sensor code uses a 1 second moving window. The accelerometer data is stored in Cartesian coordinates.

```
// code made by Gijs van Rhijn, 2019
// This code is based on code by Jeff Roberg
/* ============================================
  I2Cdev device library code is placed under the MIT license
  Copyright (c) 2012 Jeff Rowberg

  Permission is hereby granted, free of charge, to any person obtaining a copy
  of this software and associated documentation files (the "Software"), to deal
  in the Software without restriction, including without limitation the rights
  to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
  copies of the Software, and to permit persons to whom the Software is
  furnished to do so, subject to the following conditions:

  The above copyright notice and this permission notice shall be included in
  all copies or substantial portions of the Software.

  THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
  IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
  FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
  AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
  LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
  OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
  THE SOFTWARE.
  ===============================================
*/
// The code for the Adafruit TCA9548A multiplexer is based on a tutorial by Adafruit
   : https://learn.adafruit.com/adafruit-tca9548a-1-to-8-i2c-multiplexer-
  breakout/wiring-and-test

// I2Cdev and MPU6050 must be installed as libraries, or else the .cpp/.h files
// for both classes must be in the include path of your project
#include "I2Cdev.h"

#include "MPU6050_6Axis_MotionApps20.h"
//#include "MPU6050.h" // not necessary if using MotionApps include file

// Arduino Wire library is required if I2Cdev I2CDEV_ARDUINO_WIRE implementation
// is used in I2Cdev.h
#if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
#include "Wire.h"
#endif

// class default I2C address is 0x68
// specific I2C addresses may be passed as a parameter here
// AD0 low = 0x68 (default for SparkFun breakout and InvenSense evaluation board)
// AD0 high = 0x69
MPU6050 mpu;
MPU6050 mpu2;
//MPU6050 mpu(0x68); // <-- use for AD0 high

/* =========================================================================
   NOTE: In addition to connection 3.3v, GND, SDA, and SCL, this sketch
```

```
51.    depends on the MPU-6050's INT pin being connected to the Arduino's
52.    external interrupt #0 pin. On the Arduino Uno and Mega 2560, this is
53.    digital I/O pin 2.
54.    ============================================================================ */
55.
56. /* ============================================================================
57.    NOTE: Arduino v1.0.1 with the Leonardo board generates a compile error
58.    when using Serial.write(buf, len). The Teapot output uses this method.
59.    The solution requires a modification to the Arduino USBAPI.h file, which
60.    is fortunately simple, but annoying. This will be fixed in the next IDE
61.    release. For more info, see these links:
62.
63.    http://arduino.cc/forum/index.php/topic,109987.0.html
64.    http://code.google.com/p/arduino/issues/detail?id=958
65.    ============================================================================ */
66.
67.
68.
69. // uncomment "OUTPUT_READABLE_QUATERNION" if you want to see the actual
70. // quaternion components in a [w, x, y, z] format (not best for parsing
71. // on a remote host such as Processing or something though)
72. //#define OUTPUT_READABLE_QUATERNION
73.
74. // uncomment "OUTPUT_READABLE_EULER" if you want to see Euler angles
75. // (in degrees) calculated from the quaternions coming from the FIFO.
76. // Note that Euler angles suffer from gimbal lock (for more info, see
77. // http://en.wikipedia.org/wiki/Gimbal_lock)
78. //#define OUTPUT_READABLE_EULER
79.
80. // uncomment "OUTPUT_READABLE_YAWPITCHROLL" if you want to see the yaw/
81. // pitch/roll angles (in degrees) calculated from the quaternions coming
82. // from the FIFO. Note this also requires gravity vector calculations.
83. // Also note that yaw/pitch/roll angles suffer from gimbal lock (for
84. // more info, see: http://en.wikipedia.org/wiki/Gimbal_lock)
85. //#define OUTPUT_READABLE_YAWPITCHROLL //this one for gyro
86.
87. // uncomment "OUTPUT_READABLE_REALACCEL" if you want to see acceleration
88. // components with gravity removed. This acceleration reference frame is
89. // not compensated for orientation, so +X is always +X according to the
90. // sensor, just without the effects of gravity. If you want acceleration
91. // compensated for orientation, us OUTPUT_READABLE_WORLDACCEL instead.
92. //#define OUTPUT_READABLE_REALACCEL
93.
94. // uncomment "OUTPUT_READABLE_WORLDACCEL" if you want to see acceleration
95. // components with gravity removed and adjusted for the world frame of
96. // reference (yaw is relative to initial orientation, since no magnetometer
97. // is present in this case). Could be quite handy in some cases.
98. //#define OUTPUT_READABLE_WORLDACCEL // this one for acccel
99.
100.       // uncomment "OUTPUT_TEAPOT" if you want output that matches the
101.       // format used for the InvenSense teapot demo
102.       //#define OUTPUT_TEAPOT
103.
104.
105.
106.       #define LED_PIN 13 // (Arduino is 13, Teensy is 11, Teensy++ is 6)
107.       bool blinkState = false;
108.
109.       // MPU control/status vars
110.       bool dmpReady = false;  // set true if DMP init was successful
111.       uint8_t mpuIntStatus;   // holds actual interrupt status byte from MPU
112.       uint8_t devStatus;      // return status after each device operation (0 = suc
    cess, !0 = error)
113.       uint16_t packetSize;    // expected DMP packet size (default is 42 bytes)
114.       uint16_t fifoCount;     // count of all bytes currently in FIFO
115.       uint8_t fifoBuffer[64]; // FIFO storage buffer
```

```
116.
117.       // orientation/motion vars
118.       Quaternion q;           // [w, x, y, z]         quaternion container
119.       VectorInt16 aa;         // [x, y, z]            accel sensor measurements
120.       VectorInt16 aaReal;     // [x, y, z]            gravity-
    free accel sensor measurements
121.       VectorInt16 aaWorld;    // [x, y, z]            world-
    frame accel sensor measurements
122.       VectorFloat gravity;    // [x, y, z]            gravity vector
123.       float euler[3];         // [psi, theta, phi]    Euler angle container
124.       float ypr[3];           // [yaw, pitch, roll]   yaw/pitch/roll container and
    gravity vector
125.
126.       // packet structure for InvenSense teapot demo
127.       uint8_t teapotPacket[14] = { '$', 0x02, 0, 0, 0, 0, 0, 0, 0, 0, 0x00, 0x00, '
    \r', '\n' };
128.
129.       // ================================================================
130.       // ===               VARIABLES NEEDED FOR MULTIPLE MPUs          ===
131.       // ================================================================
132.       //#include "Arduino.h"
133.       //#include "class-MpuData.h";
134.       #include <MpuData.h>
135.       const byte gyroCount = 6; //amount of gyros connected
136.       byte gyro = 0; //current gyro being read by the arduino
137.       int mpuData[gyroCount][6] = {0}; //2 dimensional array containing the data fo
    r each gyro (yaw, pitch roll, x, y, z)
138.       int mpuDataAdded[gyroCount][6] = {0}; //2 dimensional array containing the ad
    ded up data for each gyro (yaw, pitch roll, x, y, z)
139.       int stretchDataAdded[2];
140.       long startTime = 0; //the moment when the current interval started
141.       static int interval = 1000; //the duration of the moving interval
142.       int numOfValues = 0; // the number of values that have been added together du
    ring the current interval
143.       bool shouldSend;
144.
145.       // ================================================================
146.       // ===               VARIABLES NEEDED FOR SERIAL COM             ===
147.       // ================================================================
148.
149.       bool snd = false;
150.
151.
152.       // ================================================================
153.       // ===                INTERRUPT DETECTION ROUTINE                ===
154.       // ================================================================
155.
156.       volatile bool mpuInterrupt = false;     // indicates whether MPU interrupt pi
    n has gone high
157.       void dmpDataReady() {
158.         mpuInterrupt = true;
159.       }
160.
161.
162.
163.       // ================================================================
164.       // ===                      INITIAL SETUP                       ===
165.       // ================================================================
166.
167.       void setup() {
168.         Serial.begin(115200);
169.         // join I2C bus (I2Cdev library doesn't do this automatically)
170.       #if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
171.         Wire.begin();
172.         TWBR = 24; // 400kHz I2C clock (200kHz if CPU is 8MHz)
173.       #elif I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_FASTWIRE
```

```
174.          Fastwire::setup(400, true);
175.        #endif
176.          tcaselect(0); // select the right gyro
177.
178.          // initialize serial communication
179.          // (115200 chosen because it is required for Teapot Demo output, but it's
180.          // really up to you depending on your project)
181.
182.          while (!Serial); // wait for Leonardo enumeration, others continue immediat
    ely
183.
184.          // NOTE: 8MHz or slower host processors, like the Teensy @ 3.3v or Ardunio

185.          // Pro Mini running at 3.3v, cannot handle this baud rate reliably due to
186.          // the baud timing being too misaligned with processor ticks. You must use

187.          // 38400 or slower in these cases, or use some kind of external separate
188.          // crystal solution for the UART timer.
189.
190.          // initialize device in loop
191.          Serial.println(F("Initializing I2C devices..."));
192.          for (int i = 0; i < gyroCount; i++) {
193.            tcaselect(i); // select the right gyro
194.            mpu.initialize();
195.            Serial.print("initializing: ");
196.            Serial.println(i);
197.          }
198.          tcaselect(0); // select the right gyro
199.
200.          // verify connection in loop
201.          Serial.println(F("Testing device connections..."));
202.          for (int i = 0; i < gyroCount; i++) {
203.            tcaselect(i); // select the right gyro
204.            Serial.println(mpu.testConnection() ? F("MPU6050 connection successful")
    : F("MPU6050 connection failed"));
205.          }
206.          tcaselect(0); // select the right gyro
207.
208.          // wait for ready
209.          Serial.println(F("\nSend any character to begin DMP programming and demo: "
    ));
210.          while (Serial.available() && Serial.read()); // empty buffer
211.          while (!Serial.available());                 // wait for data
212.          while (Serial.available() && Serial.read()); // empty buffer again
213.
214.          // load and configure the DMP in loop
215.          Serial.println(F("Initializing DMP..."));
216.          for (int i = 0; i < gyroCount; i++) {
217.            tcaselect(i); // select the right gyro
218.            devStatus = mpu.dmpInitialize();
219.          }
220.          tcaselect(0); // select the right gyro
221.
222.          // supply your own gyro offsets here, scaled for min sensitivity
223.          mpu.setXGyroOffset(220);
224.          mpu.setYGyroOffset(76);
225.          mpu.setZGyroOffset(-85);
226.          mpu.setZAccelOffset(1788); // 1688 factory default for my test chip
227.
228.
229.          // make sure it worked (returns 0 if so)
230.          if (devStatus == 0) {
231.            // turn on the DMP, now that it's ready
232.            Serial.println(F("Enabling DMP..."));
233.            for (int i = 0; i < gyroCount; i++) {
234.              tcaselect(i); // select the right gyro
```

```
235.            mpu.setDMPEnabled(true);
236.          }
237.          tcaselect(0); // select the right gyro
238.
239.          // enable Arduino interrupt detection
240.          Serial.println(F("Enabling interrupt detection (Arduino external interrup
    t 0)..."));
241.          attachInterrupt(0, dmpDataReady, RISING);
242.          mpuIntStatus = mpu.getIntStatus();
243.
244.          // set our DMP Ready flag so the main loop() function knows it's okay to
    use it
245.          Serial.println(F("DMP ready! Waiting for first interrupt..."));
246.          dmpReady = true;
247.          //
248.          // get expected DMP packet size for later comparison
249.          packetSize = mpu.dmpGetFIFOPacketSize();
250.        } else {
251.          // ERROR!
252.          // 1 = initial memory load failed
253.          // 2 = DMP configuration updates failed
254.          // (if it's going to break, usually the code will be 1)
255.          Serial.print(F("DMP Initialization failed (code "));
256.          Serial.print(devStatus);
257.          Serial.println(F(")"));
258.        }
259.
260.        // configure LED for output
261.        pinMode(LED_PIN, OUTPUT);
262.
263.      }
264.
265.      void tcaselect(uint8_t i) {
266.        if (i > 7) return;
267.
268.        Wire.beginTransmission(0x70);
269.        Wire.write(1 << i);
270.        Wire.endTransmission();
271.      }
272.
273.      // ================================================================
274.      // ===                      MAIN PROGRAM LOOP                     ===
275.      // ================================================================
276.
277.      void loop() {
278.        gyro = gyro % gyroCount;
279.        checkSnd(); //check if there is a message from the serial port
280.
281.        //  if (gyro == 0) {
282.        //    gyro = 1;
283.        //  }
284.        //  else gyro = 0;
285.
286.        tcaselect(gyro); // select the right gyro
287.
288.        // if programming failed, don't try to do anything
289.        if (!dmpReady) return;
290.
291.        // wait for MPU interrupt or extra packet(s) available
292.        while (!mpuInterrupt && fifoCount < packetSize) {
293.          // other program behavior stuff here
294.          // .
295.          // .
296.          // .
297.          // if you are really paranoid you can frequently test in between other
298.          // stuff to see if mpuInterrupt is true, and if so, "break;" from the
```

```
299.           // while() loop to immediately process the MPU data
300.           // .
301.           // .
302.           // .
303.         }
304.
305.         // reset interrupt flag and get INT_STATUS byte
306.         mpuInterrupt = false;
307.         mpuIntStatus = mpu.getIntStatus();
308.         // get current FIFO count
309.         fifoCount = mpu.getFIFOCount();
310.         //  Serial.println(fifoCount);
311.
312.         // check for overflow (this should never happen unless our code is too inef
    ficient)
313.         if ((mpuIntStatus & 0x10) || fifoCount == 1024) {
314.           // reset so we can continue cleanly
315.           mpu.resetFIFO();
316.           //    Serial.println(F("FIFO overflow!"));
317.
318.           // otherwise, check for DMP data ready interrupt (this should happen freq
    uently)
319.         } else if (mpuIntStatus & 0x02) {
320.           // wait for correct available data length, should be a VERY short wait
321.           while (fifoCount < packetSize) fifoCount = mpu.getFIFOCount();
322.
323.           // read a packet from FIFO
324.           mpu.getFIFOBytes(fifoBuffer, packetSize);
325.
326.           // track FIFO count here in case there is > 1 packet available
327.           // (this lets us immediately read more without waiting for an interrupt)

328.           fifoCount -= packetSize;
329.
330.           //    Serial.print("gyro: ");
331.           //    Serial.println(gyro);
332.
333.           saveData();
334.
335.           //start the interval
336.           if (snd) {
337.             snd = false;
338.             startTime = millis();
339.             shouldSend = true;
340.       //      Serial.println(millis());
341.       //      Serial.println(startTime);
342.
343.             //reset all the variables for the next loop
344.             numOfValues = 0;
345.             for (int g = 0; g < gyroCount; g++) {
346.               mpuDataAdded[g][0] = 0;
347.               mpuDataAdded[g][1] = 0;
348.               mpuDataAdded[g][2] = 0;
349.               mpuDataAdded[g][3] = 0;
350.               mpuDataAdded[g][4] = 0;
351.               mpuDataAdded[g][5] = 0;
352.             }
353.
354.             stretchDataAdded[0] = 0;
355.             stretchDataAdded[1] = 0;
356.
357.           }
358.
359.           //add up all the data in the interval
360.           if (shouldSend) {
361.             numOfValues++;
```

```
362.              for (int g = 0; g < gyroCount; g++) {
363.                mpuDataAdded[g][0] += mpuData[g][0];
364.                mpuDataAdded[g][1] += mpuData[g][1];
365.                mpuDataAdded[g][2] += mpuData[g][2];
366.                mpuDataAdded[g][3] += mpuData[g][3];
367.                mpuDataAdded[g][4] += mpuData[g][4];
368.                mpuDataAdded[g][5] += mpuData[g][5];
369.              }
370.
371.              stretchDataAdded[0] += analogRead(A3);
372.              stretchDataAdded[1] += analogRead(A6);
373.            }
374.
375.            //end the intervall and send the data
376.            if (interval <= millis() - startTime && shouldSend) {
377.        //      Serial.println(millis() - startTime);
378.              shouldSend = false;
379.              for (int g = 0; g < gyroCount; g++) {
380.                Serial.print(mpuDataAdded[g][0] / numOfValues);
381.                Serial.print(";");
382.                Serial.print(mpuDataAdded[g][1] / numOfValues);
383.                Serial.print(";");
384.                Serial.print(mpuDataAdded[g][2] / numOfValues);
385.                Serial.print(";");
386.                Serial.print(mpuDataAdded[g][3] / numOfValues);
387.                Serial.print(";");
388.                Serial.print(mpuDataAdded[g][4] / numOfValues);
389.                Serial.print(";");
390.                Serial.print(mpuDataAdded[g][5] / numOfValues);
391.                Serial.print(";");
392.              }
393.              Serial.print(stretchDataAdded[0] / numOfValues);
394.              Serial.print(";");
395.              Serial.print(stretchDataAdded[1] / numOfValues);
396.              Serial.println("");
397.
398.              //reset all the variables for the next loop
399.              numOfValues = 0;
400.              for (int g = 0; g < gyroCount; g++) {
401.                mpuDataAdded[g][0] = 0;
402.                mpuDataAdded[g][1] = 0;
403.                mpuDataAdded[g][2] = 0;
404.                mpuDataAdded[g][3] = 0;
405.                mpuDataAdded[g][4] = 0;
406.                mpuDataAdded[g][5] = 0;
407.              }
408.
409.              stretchDataAdded[0] = 0;
410.              stretchDataAdded[1] = 0;
411.
412.            }
413.          }
414.          gyro++;
415.        }
416.
417.        void checkSnd() {
418.          if (Serial.available()) {
419.            Serial.read();
420.            snd = true;
421.          }
422.        }
423.
424.        //void sendMsg() {
425.        //  //implement
426.        //}
427.
```

```
428.      void saveData() {
429.        // display Euler angles in degrees
430.        mpu.dmpGetQuaternion(&q, fifoBuffer);
431.        mpu.dmpGetGravity(&gravity, &q);
432.        mpu.dmpGetYawPitchRoll(ypr, &q, &gravity);
433.
434.        int yaw = ypr[0] * 180 / M_PI;
435.        int pitch = ypr[1] * 180 / M_PI;
436.        int roll = ypr[2] * 180 / M_PI;
437.        mpuData [gyro] [0] = yaw;
438.        mpuData [gyro] [1] = pitch;
439.        mpuData [gyro] [2] = roll;
440.
441.        // display initial world-frame acceleration, adjusted to remove gravity
442.        // and rotated based on known orientation from quaternion
443.        mpu.dmpGetQuaternion(&q, fifoBuffer);
444.        mpu.dmpGetAccel(&aa, fifoBuffer);
445.        mpu.dmpGetGravity(&gravity, &q);
446.        mpu.dmpGetLinearAccel(&aaReal, &aa, &gravity);
447.        mpu.dmpGetLinearAccelInWorld(&aaWorld, &aaReal, &q);
448.        //  Serial.print("aworld\t");
449.        //  Serial.print(aaWorld.x);
450.        //  Serial.print("\t");
451.        //  Serial.print(aaWorld.y);
452.        //  Serial.print("\t");
453.        //  Serial.println(aaWorld.z);
454.
455.        int x = aaWorld.x;
456.        int y = aaWorld.y;
457.        int z = aaWorld.z;
458.        mpuData [gyro] [3] = x;
459.        mpuData [gyro] [4] = y;
460.        mpuData [gyro] [5] = z;
461.      }
```

Appendix B-III, third version of the Arduino code.

This version of the sensor code uses a 1 second moving window. The accelerometer data is stored in normalized spherical coordinates.

```
1.  // code made by Gijs van Rhijn, 2019
2.  // This code is based on code by Jeff Roberg
3.  /* ============================================
4.    I2Cdev device library code is placed under the MIT license
5.    Copyright (c) 2012 Jeff Rowberg
6.
7.    Permission is hereby granted, free of charge, to any person obtaining a copy
8.    of this software and associated documentation files (the "Software"), to deal
9.    in the Software without restriction, including without limitation the rights
10.   to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
11.   copies of the Software, and to permit persons to whom the Software is
12.   furnished to do so, subject to the following conditions:
13.
14.   The above copyright notice and this permission notice shall be included in
15.   all copies or substantial portions of the Software.
16.
17.   THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
18.   IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
19.   FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
20.   AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
21.   LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
22.   OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
23.   THE SOFTWARE.
```

```
  ============================================
*/
// The code for the Adafruit TCA9548A multiplexer is based on a tutorial by Adafruit: https://learn.adafruit.com/adafruit-tca9548a-1-to-8-i2c-multiplexer-breakout/wiring-and-test

// I2Cdev and MPU6050 must be installed as libraries, or else the .cpp/.h files
// for both classes must be in the include path of your project
#include "I2Cdev.h"

#include "MPU6050_6Axis_MotionApps20.h"
//#include "MPU6050.h" // not necessary if using MotionApps include file

// Arduino Wire library is required if I2Cdev I2CDEV_ARDUINO_WIRE implementation
// is used in I2Cdev.h
#if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
#include "Wire.h"
#endif

// class default I2C address is 0x68
// specific I2C addresses may be passed as a parameter here
// AD0 low = 0x68 (default for SparkFun breakout and InvenSense evaluation board)
// AD0 high = 0x69
MPU6050 mpu;
MPU6050 mpu2;
//MPU6050 mpu(0x68); // <-- use for AD0 high

//include for spherical coordinates
#include <math.h>;

/* =========================================================================
   NOTE: In addition to connection 3.3v, GND, SDA, and SCL, this sketch
   depends on the MPU-6050's INT pin being connected to the Arduino's
   external interrupt #0 pin. On the Arduino Uno and Mega 2560, this is
   digital I/O pin 2.
   ========================================================================= */

/* =========================================================================
   NOTE: Arduino v1.0.1 with the Leonardo board generates a compile error
   when using Serial.write(buf, len). The Teapot output uses this method.
   The solution requires a modification to the Arduino USBAPI.h file, which
   is fortunately simple, but annoying. This will be fixed in the next IDE
   release. For more info, see these links:

   http://arduino.cc/forum/index.php/topic,109987.0.html
   http://code.google.com/p/arduino/issues/detail?id=958
   ========================================================================= */

// the acceleration vectors in this code use the "OUTPUT_READABLE_WORLDACCEL" calculations fromthe example MPU 6050 code:
// components with gravity removed and adjusted for the world frame of
// reference (yaw is relative to initial orientation, since no magnetometer
// is present in this case). Could be quite handy in some cases.



#define LED_PIN 13 // (Arduino is 13, Teensy is 11, Teensy++ is 6)
bool blinkState = false;

// MPU control/status vars
bool dmpReady = false;  // set true if DMP init was successful
uint8_t mpuIntStatus;   // holds actual interrupt status byte from MPU
uint8_t devStatus;      // return status after each device operation (0 = success, !0 = error)
uint16_t packetSize;    // expected DMP packet size (default is 42 bytes)
```

```
uint16_t fifoCount;     // count of all bytes currently in FIFO
uint8_t fifoBuffer[64]; // FIFO storage buffer

// orientation/motion vars
Quaternion q;           // [w, x, y, z]         quaternion container
VectorInt16 aa;         // [x, y, z]            accel sensor measurements
VectorInt16 aaReal;     // [x, y, z]            gravity-free accel sensor measurements
VectorInt16 aaWorld;    // [x, y, z]            world-frame accel sensor measurements
VectorFloat gravity;    // [x, y, z]            gravity vector
float euler[3];         // [psi, theta, phi]    Euler angle container
float ypr[3];           // [yaw, pitch, roll]   yaw/pitch/roll container and gravity vector

// packet structure for InvenSense teapot demo
uint8_t teapotPacket[14] = { '$', 0x02, 0, 0, 0, 0, 0, 0, 0, 0, 0x00, 0x00, '\r', '\n' };

      // ================================================================
      // ===               VARIABLES NEEDED FOR MULTIPLE MPUs          ===
      // ================================================================
      //#include "Arduino.h"
      //#include "class-MpuData.h";
      #include <MpuData.h>
      const byte gyroCount = 6; //amount of gyros connected
      byte gyro = 0; //current gyro being read by the arduino
      float mpuData[gyroCount][6] = {0}; //2 dimensional array containing the data for each gyro (yaw, pitch roll, x, y, z)
      float mpuDataAdded[gyroCount][6] = {0}; //2 dimensional array containing the added up data for each gyro (yaw, pitch roll, x, y, z)
      int stretchDataAdded[2];
      long startTime = 0; //the moment when the current interval started
      static int interval = 1000; //the duration of the moving interval
      int numOfValues = 0; // the number of values that have been added together during the current interval
      bool shouldSend;

      // ================================================================
      // ===               VARIABLES NEEDED FOR SERIAL COM             ===
      // ================================================================

      bool snd = false;


      // ================================================================
      // ===                INTERRUPT DETECTION ROUTINE                ===
      // ================================================================

      volatile bool mpuInterrupt = false;     // indicates whether MPU interrupt pin has gone high
      void dmpDataReady() {
        mpuInterrupt = true;
      }


      // ================================================================
      // ===                      INITIAL SETUP                       ===
      // ================================================================

      void setup() {
        Serial.begin(115200);
        // join I2C bus (I2Cdev library doesn't do this automatically)
      #if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
        Wire.begin();
```

```
144.          TWBR = 24; // 400kHz I2C clock (200kHz if CPU is 8MHz)
145.        #elif I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_FASTWIRE
146.          Fastwire::setup(400, true);
147.        #endif
148.          tcaselect(0); // select the right gyro
149.
150.          // initialize serial communication
151.          // (115200 chosen because it is required for Teapot Demo output, but it's
152.          // really up to you depending on your project)
153.
154.          while (!Serial); // wait for Leonardo enumeration, others continue immediat
    ely
155.
156.          // NOTE: 8MHz or slower host processors, like the Teensy @ 3.3v or Ardunio

157.          // Pro Mini running at 3.3v, cannot handle this baud rate reliably due to
158.          // the baud timing being too misaligned with processor ticks. You must use

159.          // 38400 or slower in these cases, or use some kind of external separate
160.          // crystal solution for the UART timer.
161.
162.          // initialize device in loop
163.          Serial.println(F("Initializing I2C devices..."));
164.          for (int i = 0; i < gyroCount; i++) {
165.            tcaselect(i); // select the right gyro
166.            mpu.initialize();
167.            Serial.print("initializing: ");
168.            Serial.println(i);
169.          }
170.          tcaselect(0); // select the right gyro
171.
172.          // verify connection in loop
173.          Serial.println(F("Testing device connections..."));
174.          for (int i = 0; i < gyroCount; i++) {
175.            tcaselect(i); // select the right gyro
176.            Serial.println(mpu.testConnection() ? F("MPU6050 connection successful")
    : F("MPU6050 connection failed"));
177.          }
178.          tcaselect(0); // select the right gyro
179.
180.          // wait for ready
181.          Serial.println(F("\nSend any character to begin DMP programming and demo: "
    ));
182.          while (Serial.available() && Serial.read()); // empty buffer
183.          while (!Serial.available());                 // wait for data
184.          while (Serial.available() && Serial.read()); // empty buffer again
185.
186.          // load and configure the DMP in loop
187.          Serial.println(F("Initializing DMP..."));
188.          for (int i = 0; i < gyroCount; i++) {
189.            tcaselect(i); // select the right gyro
190.            devStatus = mpu.dmpInitialize();
191.          }
192.          tcaselect(0); // select the right gyro
193.
194.          // supply your own gyro offsets here, scaled for min sensitivity
195.          mpu.setXGyroOffset(220);
196.          mpu.setYGyroOffset(76);
197.          mpu.setZGyroOffset(-85);
198.          mpu.setZAccelOffset(1788); // 1688 factory default for my test chip
199.
200.
201.          // make sure it worked (returns 0 if so)
202.          if (devStatus == 0) {
203.            // turn on the DMP, now that it's ready
204.            Serial.println(F("Enabling DMP..."));
```

```
205.          for (int i = 0; i < gyroCount; i++) {
206.            tcaselect(i); // select the right gyro
207.            mpu.setDMPEnabled(true);
208.          }
209.          tcaselect(0); // select the right gyro
210.
211.          // enable Arduino interrupt detection
212.          Serial.println(F("Enabling interrupt detection (Arduino external interrup
    t 0)..."));
213.          attachInterrupt(0, dmpDataReady, RISING);
214.          mpuIntStatus = mpu.getIntStatus();
215.
216.          // set our DMP Ready flag so the main loop() function knows it's okay to
    use it
217.          Serial.println(F("DMP ready! Waiting for first interrupt..."));
218.          dmpReady = true;
219.          //
220.          // get expected DMP packet size for later comparison
221.          packetSize = mpu.dmpGetFIFOPacketSize();
222.        } else {
223.          // ERROR!
224.          // 1 = initial memory load failed
225.          // 2 = DMP configuration updates failed
226.          // (if it's going to break, usually the code will be 1)
227.          Serial.print(F("DMP Initialization failed (code "));
228.          Serial.print(devStatus);
229.          Serial.println(F(")"));
230.        }
231.
232.        // configure LED for output
233.        pinMode(LED_PIN, OUTPUT);
234.
235.      }
236.
237.      void tcaselect(uint8_t i) {
238.        if (i > 7) return;
239.
240.        Wire.beginTransmission(0x70);
241.        Wire.write(1 << i);
242.        Wire.endTransmission();
243.      }
244.
245.      // ================================================================
246.      // ===                      MAIN PROGRAM LOOP                     ===
247.      // ================================================================
248.
249.      void loop() {
250.        gyro = gyro % gyroCount;
251.        checkSnd(); //check if there is a message from the serial port
252.
253.        //  if (gyro == 0) {
254.        //    gyro = 1;
255.        //  }
256.        //  else gyro = 0;
257.
258.        tcaselect(gyro); // select the right gyro
259.
260.        // if programming failed, don't try to do anything
261.        if (!dmpReady) return;
262.
263.        // wait for MPU interrupt or extra packet(s) available
264.        while (!mpuInterrupt && fifoCount < packetSize) {
265.          // other program behavior stuff here
266.          // .
267.          // .
268.          // .
```

```
269.           // if you are really paranoid you can frequently test in between other
270.           // stuff to see if mpuInterrupt is true, and if so, "break;" from the
271.           // while() loop to immediately process the MPU data
272.           // .
273.           // .
274.           // .
275.         }
276.
277.         // reset interrupt flag and get INT_STATUS byte
278.         mpuInterrupt = false;
279.         mpuIntStatus = mpu.getIntStatus();
280.         // get current FIFO count
281.         fifoCount = mpu.getFIFOCount();
282.         //  Serial.println(fifoCount);
283.
284.         // check for overflow (this should never happen unless our code is too inef
    ficient)
285.         if ((mpuIntStatus & 0x10) || fifoCount == 1024) {
286.           // reset so we can continue cleanly
287.           mpu.resetFIFO();
288.           //    Serial.println(F("FIFO overflow!"));
289.
290.           // otherwise, check for DMP data ready interrupt (this should happen freq
    uently)
291.         } else if (mpuIntStatus & 0x02) {
292.           // wait for correct available data length, should be a VERY short wait
293.           while (fifoCount < packetSize) fifoCount = mpu.getFIFOCount();
294.
295.           // read a packet from FIFO
296.           mpu.getFIFOBytes(fifoBuffer, packetSize);
297.
298.           // track FIFO count here in case there is > 1 packet available
299.           // (this lets us immediately read more without waiting for an interrupt)

300.           fifoCount -= packetSize;
301.
302.           //    Serial.print("gyro: ");
303.           //    Serial.println(gyro);
304.
305.           saveData();
306.
307.           //start the interval
308.           if (snd) {
309.             snd = false;
310.             startTime = millis();
311.             shouldSend = true;
312.             //      Serial.println(millis());
313.             //      Serial.println(startTime);
314.
315.             //reset all the variables for the next loop
316.             numOfValues = 0;
317.             for (int g = 0; g < gyroCount; g++) {
318.               mpuDataAdded[g][0] = 0;
319.               mpuDataAdded[g][1] = 0;
320.               mpuDataAdded[g][2] = 0;
321.               mpuDataAdded[g][3] = 0;
322.               mpuDataAdded[g][4] = 0;
323.               mpuDataAdded[g][5] = 0;
324.             }
325.
326.             stretchDataAdded[0] = 0;
327.             stretchDataAdded[1] = 0;
328.
329.           }
330.
331.           //add up all the data in the interval
```

```
332.           if (shouldSend) {
333.             numOfValues++;
334.             for (int g = 0; g < gyroCount; g++) {
335.               mpuDataAdded[g][0] += mpuData[g][0];
336.               mpuDataAdded[g][1] += mpuData[g][1];
337.               mpuDataAdded[g][2] += mpuData[g][2];
338.               mpuDataAdded[g][3] += mpuData[g][3];
339.               mpuDataAdded[g][4] += mpuData[g][4];
340.               mpuDataAdded[g][5] += mpuData[g][5];
341.             }
342.
343.             stretchDataAdded[0] += analogRead(A3);
344.             stretchDataAdded[1] += analogRead(A6);
345.           }
346.
347.           //end the intervall and send the data
348.           if (interval <= millis() - startTime && shouldSend) {
349.             shouldSend = false;
350.             for (int g = 0; g < gyroCount; g++) {
351.               float x = mpuDataAdded[g][3] / numOfValues;
352.               float y = mpuDataAdded[g][4] / numOfValues;
353.               float z = mpuDataAdded[g][5] / numOfValues;
354.               float magnitude = sqrt((x * x) + (y * y) + (z * z)); //calc the magni
    tude of the vector
355.               x = x / magnitude;
356.               y = y / magnitude;
357.               z = z / magnitude;
358.
359.               double O = atan2 (y,x); //calc angle between the x and y coord
360.               double P = atan2 (x,z); // calc angle between the x and z coord
361.               //the r of every coord is 1
362.
363.               Serial.print(mpuDataAdded[g][0] / numOfValues);
364.               Serial.print(";");
365.               Serial.print(mpuDataAdded[g][1] / numOfValues);
366.               Serial.print(";");
367.               Serial.print(mpuDataAdded[g][2] / numOfValues);
368.               Serial.print(";");
369.               Serial.print(O);
370.               Serial.print(";");
371.               Serial.print(P);
372.               Serial.print(";");
373.             }
374.             Serial.print(stretchDataAdded[0] / numOfValues);
375.             Serial.print(";");
376.             Serial.print(stretchDataAdded[1] / numOfValues);
377.             Serial.println("");
378.
379.             //reset all the variables for the next loop
380.             numOfValues = 0;
381.             for (int g = 0; g < gyroCount; g++) {
382.               mpuDataAdded[g][0] = 0;
383.               mpuDataAdded[g][1] = 0;
384.               mpuDataAdded[g][2] = 0;
385.               mpuDataAdded[g][3] = 0;
386.               mpuDataAdded[g][4] = 0;
387.               mpuDataAdded[g][5] = 0;
388.             }
389.
390.             stretchDataAdded[0] = 0;
391.             stretchDataAdded[1] = 0;
392.
393.           }
394.         }
395.         gyro++;
396.       }
```

```
397.
398.       void checkSnd() {
399.         if (Serial.available()) {
400.           Serial.read();
401.           snd = true;
402.         }
403.       }
404.
405.       //void sendMsg() {
406.       //  //implement
407.       //}
408.
409.       void saveData() {
410.         // display Euler angles in degrees
411.         mpu.dmpGetQuaternion(&q, fifoBuffer);
412.         mpu.dmpGetGravity(&gravity, &q);
413.         mpu.dmpGetYawPitchRoll(ypr, &q, &gravity);
414.
415.         int yaw = ypr[0] * 180 / M_PI;
416.         int pitch = ypr[1] * 180 / M_PI;
417.         int roll = ypr[2] * 180 / M_PI;
418.         mpuData [gyro] [0] = yaw;
419.         mpuData [gyro] [1] = pitch;
420.         mpuData [gyro] [2] = roll;
421.
422.         // display initial world-frame acceleration, adjusted to remove gravity
423.         // and rotated based on known orientation from quaternion
424.         mpu.dmpGetQuaternion(&q, fifoBuffer);
425.         mpu.dmpGetAccel(&aa, fifoBuffer);
426.         mpu.dmpGetGravity(&gravity, &q);
427.         mpu.dmpGetLinearAccel(&aaReal, &aa, &gravity);
428.         mpu.dmpGetLinearAccelInWorld(&aaWorld, &aaReal, &q);
429.         //  Serial.print("aworld\t");
430.         //  Serial.print(aaWorld.x);
431.         //  Serial.print("\t");
432.         //  Serial.print(aaWorld.y);
433.         //  Serial.print("\t");
434.         //  Serial.println(aaWorld.z);
435.
436.         long x = aaWorld.x;
437.         long y = aaWorld.y;
438.         long z = aaWorld.z;
439.
440.         float magnitude = sqrt((x * x) + (y * y) + (z * z)); //calc the magnitude o
    f the vector
441.
442.         //normalize the 3 axis of the vector
443.         float normX = x / magnitude;
444.         float normY = y / magnitude;
445.         float normZ = z / magnitude;
446.
447.         mpuData [gyro] [3] = normX;
448.         mpuData [gyro] [4] = normY;
449.         mpuData [gyro] [5] = normZ;
450.       }
```

Appendix C, checklist ethical approval

Checklist for the principal researcher when submitting a request to the EC or the EC member for an assessment of the ethical permissibility of a research proposal

1. General

1. Title of the project: *Finding an efficient way to measure and evaluate sitting posture using a combination of body sensors placed on the body and machine learning*
2. Principal researcher (with doctoral research also a professor): Angelika Mader
3. Researchers/research assistants (PhD students, students etc. where known): Gijs van Rhijn
4. Department responsible for the research: HMI
5. Location where research will be conducted: at people's homes, on university grounds.
6. Short description of the project (about 100 words):

The goal of the project is to test the effectiveness of different sensors and machine learning techniques with regards to measuring posture quality. Classification machine learning techniques will be used to determine whether someone is sitting correctly or incorrectly. Variable importance measures will be used to try and determine what sensors and sensor placement is most useful. The sensors that will be used are: stretch sensors, gyroscopes, and accelerometers. To best determine the effectiveness of the machine learning techniques and the different sensors a reasonable amount of data needs to be gathered. For this test participants are needed.

7. Expected duration of the project and research period: the research is part of a graduation project of the creative technology bachelor program. It is expected to last until about the start of July 2019.
8.
9. Number of experimental subjects: 5 – 20, depending on how much data is needed and how many people are available.

10. EC member of the department (if available): Dennis Reidsma

2. Questions about fulfilled general requirements and conditions

1. Has this research or similar research by the department been previously submitted to the EC?
   ☐ Yes,

   X No

   If yes, what was the number allocated to it by the EC?

   Explanatory notes:

2. Is the research proposal to be considered as medical research (Also see Appendix 4)
   ☐ Yes

   X No

   ☐ Uncertain

Explanatory notes:

3. Are adult, competent subjects selected? (§3.2)
   X Yes, indicate in which of the ways named in the general requirements and conditions this is so

   ☐ No, explain

   ☐ Uncertain, explain why

   Explanatory notes: The subjects are part of group c. They are individually interesting. The subjects are chosen not because of their association with a specific group or company. Participants are chosen mainly based on availability.

4. Are the subjects completely free to participate in the research, and to withdraw from participation whenever they wish and for whatever reason? (§3.2)
   X Yes

   ☐ No, explain why not

   ☐ Uncertain, explain why

   Explanatory notes:

5. In the event that it may be necessary to screen experimental subjects in order to reduce the risks of adverse effects of the research: Will the subjects be screened? (§3.4)
   ☐ Screening is not necessary, explain why not

   ☐ Yes, explain how

   X No, explain why not

   ☐ Uncertain, explain why

   Explanatory notes: Test subjects need no special talents or abilities. If a test subject can't take on a certain posture it is left out of the tests.

6. Does the method used allow for the possibility of making an accidental diagnostic finding which the experimental subject should be informed about? (§3.6 and Appendix 4)
   X No, the method does not allow for this possibility

   ☐ Yes, and the subject has given signed assent for the method to be used

   ☐ Yes, but the subject has not given signed assent for the method to be used

   ☐ Uncertain, explain why

   Explanatory notes:

7. Are subjects briefed before participation and do they sign an informed consent beforehand in accordance with the general conditions? (§3.2, §3.3, §3.7, §3.8)
   X Yes, attach the information brochure and the form to be signed

☐ No, explain why not

☐ Uncertain, explain why

Explanatory notes:

8. Are the requirements with regard to anonymity and privacy satisfied as stipulated in (§3.8)?
   X Yes

   ☐ No, explain why not

   ☐ Uncertain, explain why

   Explanatory notes:

9. If any deception should take place, does the procedure comply with the general terms and conditions (no deception regarding risks, accurate debriefing) (§3.10)?
   X No deception takes place

   ☐ The deception which takes place complies fully with the conditions (explain)

   ☐ The deception which takes place does not comply with the conditions (explain)

   If deception does take place, attach the method of debriefing

   Explanatory notes:

10. Is it possible that after the recruitment of experimental subjects, a substantial number will withdraw from participating because, for one reason or another, the research is unpleasant? (§3.5)
    X No

    ☐ Yes, that is possible

    If yes, then attach the recruitment text paying close attention to what is stated about this in the protocol.

    Explanatory notes:

3. Questions regarding specific types of standard research

Answer the following questions based on the department to which the research belongs.

11. Does the research fall ***entirely*** under one of the descriptions of standard research as set out in the described standard research of the department? (Chapter 4)
    X Yes, go to question 12

    ☐ No, go to question 13

    ☐ Uncertain, explain what about, and go to question 13

    Explanatory notes:

12. If yes, what type of research is it? Give a more detailed specification of parts of the research which are not mentioned by name in this description (for example: What precisely are the stimuli? Or: What precisely is the task?).

The research matches the description of the non-deceitful laboratory research. Participants are asked to take on certain pre-determined sitting positions while they are being measured. The data from these sensors is gathered. No other data is gathered and no questionnaires have to be filled in. The participants do not have to make any decisions. The sensors that will be used are: accelerometers, gyroscopes, and stretch sensor.

13. If no, or if uncertain, give as complete a description as possible of the research. Refer where appropriate to the standard descriptions and indicate the differences with your research. In any case, all possible relevant data for an ethical consideration should be provided.

Appendix D, consent form

Consent Form for:

Finding an efficient way to measure and evaluate sitting posture using a combination of body sensors placed on the body and machine learning

YOU WILL BE GIVEN A COPY OF THIS INFORMED CONSENT FORM

Please tick the appropriate boxes	Yes	No
Taking part in the study		
I have read and understood the study information dated [14/04/2019], or it has been read to me. I have been able to ask questions about the study and my questions have been answered to my satisfaction.	☐	☐
I consent voluntarily to be a participant in this study and understand that I can refuse to answer questions and I can withdraw from the study at any time, without having to give a reason.	☐	☐
I understand that taking part in the study involves wearing various different sensors while taking on different sitting positions.	☐	☐
Risks associated with participating in the study		
I understand that taking part in the study involves the following risks: slight physical discomfort during the session caused by poor sitting postures.	☐	☐
Use of the information in the study		
I understand that information I provide will be used for a bachelor report that will be publicly available on completion. All data will be completely anonymous.	☐	☐
I understand that personal information collected about me that can identify me, such as your name and the time at which the data was gathered, will not be shared beyond the study team.	☐	☐
Future use and reuse of the information by others		
I give permission for the sensor that I provide to be archived in anonymized test files so it can be used for future research and learning.	☐	☐

Signatures

_____	_____	_____
Name of participant	Signature	Date

I have accurately read out the information sheet to the potential participant and, to the best of my ability, ensured that the participant understands to what they are freely consenting.

Gijs van Rhijn	_____	_____
Researcher name	Signature	Date

Study contact details for further information: Gijs van Rhijn, g.h.m.vanrhijn@student.utwente.nl

Contact Information for Questions about Your Rights as a Research Participant: g.h.m.vanrhijn@student.utwente.nl

Contact Information for Questions about Your Rights as a Research Participant
If you have questions about your rights as a research participant, or wish to obtain information, ask questions, or discuss any concerns about this study with someone other than the researcher(s), please contact the Secretary pro tem of the Ethics Committee of the department of EEMCS, mrs J Rebel-de Boer, mail: ethics-comm-ewi@utwente.nl.

Appendix E, information sheet

INFORMATION SHEET:
Finding an efficient way to measure and evaluate sitting posture using a combination of body sensors placed on the body and machine learning
14-04-2019

Introduction
You are invited to participate in a study involving the use of wearable sensors and machine learning to measure the quality of sitting posture. The goal of this study is to test the effectiveness of machine learning techniques and different types of sensors. To do this, data needs to be gathered. This data is needed to both train and test the machine learning algorithm. As a participant any consent given is voluntary. If you have questions about any of the information described in this form or anywhere else during the study, do not hesitate to contact the researcher for further explanation. As a participant, you are free to talk to anyone about the contents of this study.

Type of research intervention
To gather the data that is needed, it is necessary to conduct tests. These tests will consist of the participant wearing various sensors in different positions. Then, the participant will take on different seating postures while the sensor data is recorded. The sensor data is gathered anonymously. The seating postures that the participant will be asked to sit in will reflect realistic postures that people take on when sitting. The sensors that the participant will be wearing are as follows:

- Gyroscopes (measures rotation)
- Accelerometers (measures acceleration)
- Stretch sensors

Participation selection
This research focusses on adult people with a relatively normal posture. No special talents or skills are needed.

Voluntary Participation
This entire research is completely voluntary. No compensation will be offered to the participants, monetary or otherwise. The participant is not required to participate in the research and can opt out at any time. No reason is required for the participant to opt out of the research.

Procedures
The research will involve the collection of data through the wearable sensors. The participant will be asked to sit in various positions and sensor data belonging to each position will be stored. Comments may be added to the data about the ability of the participant to take on a certain posture. All data gathered during the experiments will be stored anonymously.

Duration
The total test for each participant is expected to take less then an hour. No follow up tests are required.

Risks
Some participants might experience discomfort when trying to take on specific sitting positions. If a participant experiences such discomfort, they are encouraged to discuss this with the researcher. When taking on the different positions the researcher will supervise to make sure the participant

does not feel overly uncomfortable. When a participant has particular difficulties taking on a seating position, it is possible to skip said position and move on to the next one to prevent any harm.

Benefits
This research benefits this research into using machine learning for posture measurement. For machine learning to work properly a significant amount of data is needed. By participating in this research, the participant helps to create and gather this data. The data gathered will be used to train and test the machine learning algorithm. The study might give insight in how to more effectively and accurately measure posture quality in the future.

Reimbursements
No reimbursement will be offered to the participants, monetary or otherwise.

Confidentiality
The only data that will be stored is the sensor data and data about what position the participant was in at the time of recording the sensor data. No information about the identity of the participant will be stored.

Sharing the Results
As mentioned before, the data gathered will be used to train a machine learning model in order to measure posture quality. This will be done as part of a bachelor thesis. If any participants are interested in the results of this study, they can contact the researcher via E-mail, WhatsApp, or in person. The final bachelor thesis will be available to the public when it is finished.

Right to Refuse or Withdraw
As mentioned before, the participant has the right to withdraw at any point during the research. The data gathered up to that point can still be used for the research, but any data gathered after cannot. No reason is needed for the participant to withdraw.

Who to Contact?
For any further contact please contact Gijs van Rhijn: g.h.m.vanrhijn@student.utwente.nl

Contact Information for Questions about Your Rights as a Research Participant
If you have questions about your rights as a research participant, or wish to obtain information, ask questions, or discuss any concerns about this study with someone other than the researcher(s), please contact the Secretary pro tem of the Ethics Committee of the department of EEMCS, mrs J Rebel-de Boer, mail: ethics-comm-ewi@utwente.nl.

How Informed Consent is described in the General Data Protection Regulation

(http://ec.europa.eu/justice/data-protection/reform/files/regulation_oj_en.pdf)

Article 4. Definitions
(11) 'consent' of the data subject means any freely given, specific, informed and unambiguous indication of the data subject's wishes by which he or she, by a statement or by a clear affirmative action, signifies agreement to the processing of personal data relating to him or her;

Article 6. Lawfulness of processing
1. Processing shall be lawful only if and to the extent that at least one of the following applies:
(a) the data subject has given consent to the processing of his or her personal data for one or more specific purposes;

Article 7. Conditions for consent
1. Where processing is based on consent, the controller shall be able to demonstrate that the data subject has consented to processing of his or her personal data.
2. If the data subject's consent is given in the context of a written declaration which also concerns other matters, the request for consent shall be presented in a manner which is clearly distinguishable from the other matters, in an intelligible and easily accessible form, using clear and plain language. Any part of such a declaration which constitutes an infringement of this Regulation shall not be binding.
3. The data subject shall have the right to withdraw his or her consent at any time. The withdrawal of consent shall not affect the lawfulness of processing based on consent before its withdrawal. Prior to giving consent, the data subject shall be informed thereof. It shall be as easy to withdraw as to give consent.
4. When assessing whether consent is freely given, utmost account shall be taken of whether, inter alia, the performance of a contract, including the provision of a service, is conditional on consent to the processing of personal data that is not necessary for the performance of that contract.

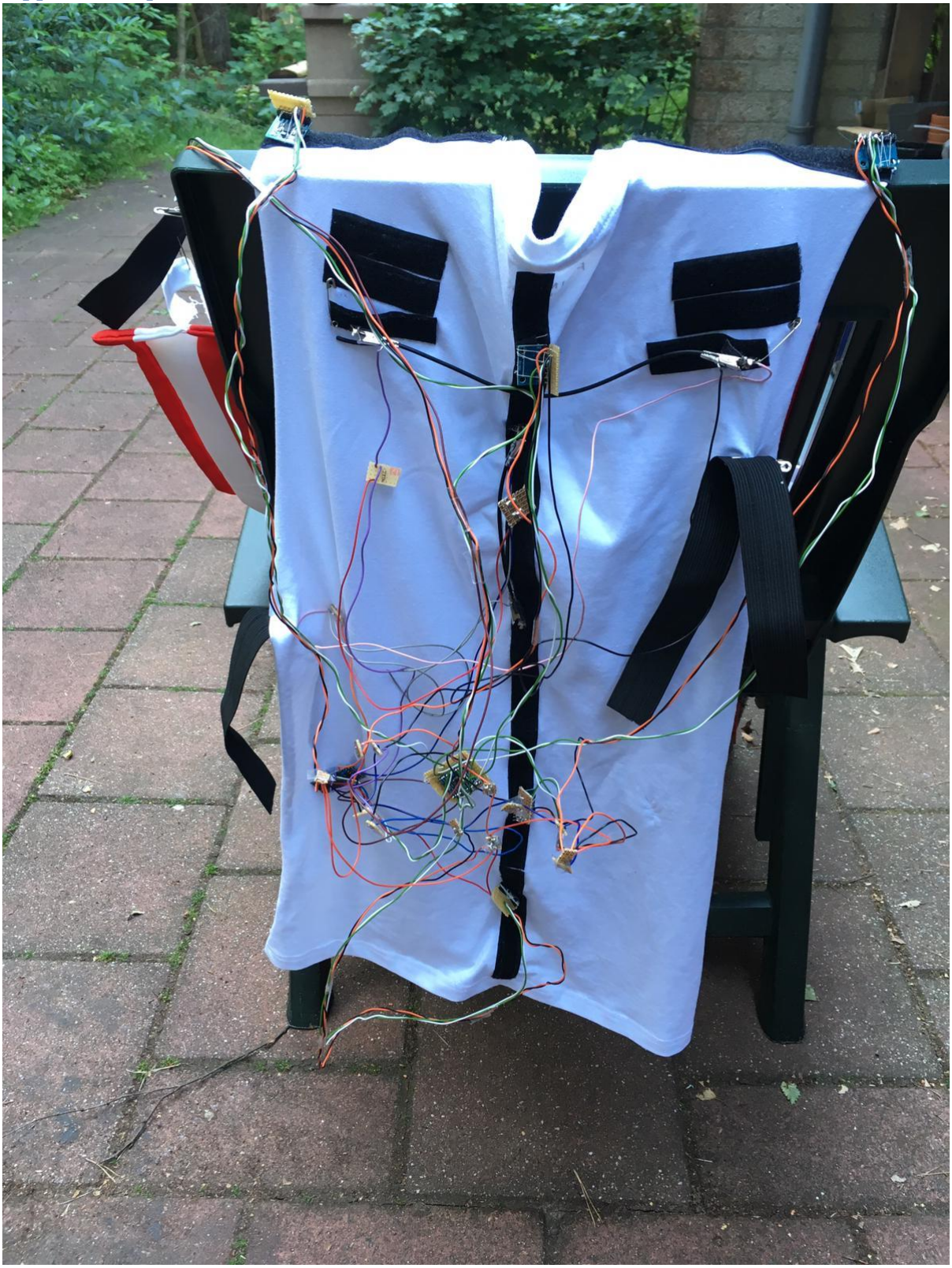
———————————————————

Recitals
(32) Consent should be given by a clear affirmative act establishing a freely given, specific, informed and unambiguous indication of the data subject's agreement to the processing of personal data relating to him or her, such as by a written statement, including by electronic means, or an oral statement. This could include ticking a box when visiting an internet website, choosing technical settings for information society services or another
statement or conduct which clearly indicates in this context the data subject's acceptance of the proposed processing of his or her personal data. Silence, pre-ticked boxes or inactivity should not therefore constitute consent. Consent should cover all processing activities carried out for the same purpose or purposes. When the processing has multiple purposes, consent should be given for all of them. If the data subject's consent is to be given following a request by electronic means, the request must be clear, concise and not unnecessarily disruptive to the use of the service for which it is provided.

(**33)** It is often not possible to fully identify the purpose of personal data processing for scientific research purposes at the time of data collection. Therefore, data subjects should be allowed to give their consent to certain areas of scientific research when in keeping with recognised ethical standards for scientific research. Data subjects should have the opportunity to give their consent only to certain areas of research or parts of research projects to the extent allowed by the intended purpose.

(**65)** A data subject should have the right to have personal data concerning him or her rectified and a 'right to be forgotten' where the retention of such data infringes this Regulation or Union or Member State law to which the controller is subject. In particular, a data subject should have the right to have his or her personal data erased and no longer processed where the personal data are no longer necessary in relation to the purposes for which they are collected or otherwise processed, where a data subject has withdrawn his or her consent or objects to the processing of personal data concerning him or her, or where the processing of his or her personal data does not otherwise comply with this Regulation. That right is relevant in particular where the data subject has given his or her consent as a child and is not fully aware of the risks involved by the processing, and later wants to remove such personal data, especially on the internet. The data subject should be able to exercise that right notwithstanding the fact that he or she is no longer a child. However, the further retention of the personal data should be lawful where it is necessary, for exercising the right of freedom of expression and information, for compliance with a legal obligation, for the performance of a task carried out in the public interest or in the
exercise of official authority vested in the controller, on the grounds of public interest in the area of public health, for archiving purposes in the public interest, scientific or historical research purposes or statistical purposes, or for the establishment, exercise or defence of legal claims.

Appendix F, picture of the real sensor suit



Appendix G, reflection report on ethical impact bachelor project

Ethical Assessment Bachelor Thesis

Gijs van Rhijn, s1721690, 09-06-2019

This paper contains an ethical assessment of my Creative Technology bachelor thesis. This assessment will be done on the basis of the ethical toolkit provided by “Ethics in practice: A toolkit” [1]. This ethical toolkit provides a structured way to assess the ethical risks and challenges of a project.

Project description

Problems relating to the back and neck are an extremely common occurrence nowadays. Many people blame this on the fact that people spend increasingly more time sitting. Sitting with an incorrect posture has been linked to various kinds of pain, most notably lower back and neck pain. It can be hard to maintain correct posture during multiple hours of work. Being able to measure posture might help people trying to improve. A posture measuring device could potentially also be used to do research about the effects of a wrong posture.

Machine learning (ML) is a promising tool to solve problems that are as complicated as posture measurement. ML techniques can find patterns in complicated, highly dimensional, data. This might give machine learning an advantage over more traditional programming techniques. By gathering data form people sitting with a correct and incorrect posture, and training a machine learning model using this data, it might be possible to create a sensor suit and machine learning model capable of determining whether one’s posture is correct or not.

Certain machine-learning techniques have one big strength in detecting variable importance. Historically, determining the importance of a specific variable has been quite complicated. The ability of ML techniques to create order in a complicated data set, allows for new possibilities in determining variable importance. Often these techniques are seemingly more accurate than more traditional techniques. Detecting variable importance can be helpful when trying to minimize the complexity of a posture sensing device, hopefully allowing the cost and complexity of the device to be reduced.

Stakeholders

To aid the identification of ethical risks, a list of stakeholders was created first.

- The researchers: the researchers involved in the project are interested in seeing the project succeed from both a personal and a professional perspective.
- Medical researchers: posture, and its effect on people’s wellbeing, is not yet fully understood. A device that is able to sense the quality of one’s posture can aide further research into this topic.
- Producers of wearable technology: producers of wearable technology are always looking for new interesting ways to give their user’s feedback into the effect their lifestyle are having on their health.

- Users of wearable technology: If the producers of wearable technology adopt a posture sensing device, the end users of the product will be using these devices in their day to day life.
- Insurance companies: insurance companies seem always interested in finding ways to improve their knowledge about the health of their customers. Posture related data, for instance from smart wearables, can be an interesting metric for them to evaluate their customers by.
- Employers: Employers can benefit immensely from keeping their workforce in good health. Back problems are known to cause significant amounts of sick leave in certain sectors of the economy. Being able to keep track of the postures of employees might be an attractive prospect to some employers.

Tool 1: ethical risk sweeping

Tool 1 of the ethical toolkit focusses on identifying the ethical risks that come with the project. In

"Ethics in practice: A toolkit" an ethical risk is defined the following way: *"Ethical risks are choices that may cause significant harm to persons or other entities with a moral status, or are likely to spark acute moral controversy for other reasons."*.

At first glance, developing a posture sensor seems quite harmless as it is a device that focusses on helping people improve their health and wellbeing. However, there are various risks associated with such technology, ranging from privacy concerns to disturbing the relationships between people and their own bodies.

The first, and arguably biggest, ethical risks relate to gathering, transmitting, and storing people's personal data. If the posture sensor technology is used to make a wearable device health in the vain of a Fitbit, it is important to keep track of how the data is stored, transmitted, and who can access it. Especially insurance agencies and employers trying to keep track of their customers'/employee's health seem like a potential source of privacy abuse.

The second risks come in the form of people's reliance on technology, and the disturbance this might have on the connection between themselves and their body. A posture sensor can tell someone weather they are sitting correctly or not according to data based on general studies and ideas of what a correct posture looks like. However, it is important to realize that human posture is a complicated, and not fully understood, subject. Like most medical problems, the best approach tends to be tailored to the individual. Of course, people with a medical condition will require specific medical advice, and should not rely solely on a posture sensor. But even people with "normal" bodies should be aware that the information given to them by a posture sensor is based on a limited set of metrics and knowledge of the average human body, making it imperfect.

The last risk relates to the increased awareness that might be created about posture quality by such a device. Smart wearables like Fitbit[2], and their marketing, have arguably increased people's awareness of the effects their lifestyle have on their long-term health. Of course, in many ways this is a good thing. People's health is extremely important to their wellbeing, so improving people's awareness and knowledge on how to improve it could have significant long-term benefits. However, awareness of, and involvement in, health related behaviour also has a tendency to separate and polarize people in society. With the help of the internet and modern fitness gadgets, different social groups have emerged based on people's relationship to fitness and diet. Causing people from different groups to look down on each other, or feel guilty about their habits and

lifestyle. These guilty feelings could have negative effects on peoples feeling of self-efficacy and belonging. Encouraging people to use and wear a posture sensor to improve and keep track of their posture quality might fuel this division even more. With people relating “straight” posture to success and wellbeing.

Tool 2: ethical pre-mortems and post-mortems

In tool 2 an ethical pre and post mortem is done to identify any systemic ethical failures that might occur. An ethical post mortem looks back at a past failure in order to learn from what happened, while an ethical pre-mortem focusses on imagining what would have gone wrong if an ethical failure had happened. In the case of this project, there are no past failures to do a post mortem on. For the reason, the ethical post mortem will be skipped. In step 4, ethical failures by third party projects will be discussed in the case-based analysis.

The ethical pre-mortem

Two potential systematic ethical risks where identified. The first risk is described already in the first step, in that there are potential privacy concerns with developing wearable health sensors than can easily be embedded in a smart wearable device. The data from these types of devices could be interesting to various third parties like insurers, but also to employers trying to keep their workforce productive. We live in a time where full privacy seems to become increasingly rare. With data being one of the most lucrative resources possessed by many large multinationals. Long term focus on the potential benefit a posture sensing device could have on society should remain at the forefront of the further development of such a device, whereas quick profits resulting in privacy violations should be avoided.

The second and thirds risk described in step one could potentially work together to create a society where people are completely detached from the reality of their posture. Similarly to how diet trends, and tracking different macronutrients, cause people to take on increasingly extreme diets, a higher awareness of posture quality combined with a disruption of the mind-body connection caused by technology might have extreme effects on people their relationships with their bodies.

Tool 3: expanding the ethical circle

Tool 3 can be described as thinking outside of your ethical box. Often, it can be hard to identify with people that do not belong to your close social circle. This can cause ethical risks to be unintentionally overlooked. Tool 3 focusses on trying to look outside of your ethical circle in order to identify ethical problems that you might not have been aware of before.

The first risk is caused by the bubble mentality. The bubble mentality described a culture where everyone is similar enough that people that do not fit within said bubble are easily forgotten. In the context of a posture sensor, this can mean the designers and developers of the posture sensor are not aware of their being people with a non-normal posture. When designing a health-oriented device, it is important to be clear about what your device is, and what it is not, able to do, and about whom it is meant to help. The developers of a posture sensing device should not fall into the trap of

trying to create a “one size fits all” solution as such a device most likely won’t give proper feedback to all its users.

The second risk comes in the form of the Friedman fallacy. The Friedman fallacy refers to a mindset where the wellbeing of a business is put above all else by its employees. This mentality can cause unethical decisions to be made in order to increase profits. Ethical risks concerning privacy violations come to mind when thinking of the Friedman fallacy. Sharing posture information with insurers and employers might be profitable, but it remains ethically questionable.

Tool 4: case-based analysis

As described before, tool 4 uses past cases to evaluate the current situation in an attempt to learn from the past. There are various other products that can be compared to a wearable posture sensor. Arguably the most obvious comparison would be the one to fitness trackers like Fitbit and the Apple watch[3]. A notable case in the history of these devices is the one involving the insurance agency John Hancock[4]. In 2015 John Hancock started to experiment with so called “interactive” policies. The customers of such policies would qualify for exclusive discounts if they decided to share their fitness tracker data with the company. Opponents of this idea described it as dystopian. The idea has been widely criticized for its breach of privacy and the control it might exercise over its users. It is not unconceivable that similar thing might happen if a posture sensor is commercialized as part of a smart wearable device. Smart health devices as a whole are regularly criticized for their potential for privacy breaches of their users. The John Hancock case is one example of such concerns becoming reality, but it is unlikely to be the last.

Tool 5: remembering the ethical benefits of creative work

Tool 5 focusses on the potential positive impact of the project. It is important to remember the benefits a project can have on the world, especially when writing a report that is focused on the risks of said project. Of course, posture sensors can have a positive effect on their users. Helping them to improve and keep track of their posture. Wearable posture sensors might be able to give people feedback on problems they are having that would normally need expensive help from a healthcare professional.

This can make personal posture feedback more widely available for the average person. Commercial wearable posture sensors are but one of the benefits of this project. Another benefit could come in the form of medical research. As mentioned before, posture and the effects it has on the human body are not yet fully understood. Being able to measure the quality of people’s posture using a compact wearable sensor could help with further research into this topic. This would help us understand the short- and long-term effects our sitting posture has on our health and wellbeing.

Tool 6: Think about the terrible people

Tool 6 focuses on abuse of the developed technology by people outside of the research team. There will always be people trying to abuse potentially good projects for their own personal gain. Again, privacy related issues come to mind. Of course, the infringing on people’s privacy by insurance agencies and employers can be considered part of this category. Because this ethical risk has been

discussed at large in previous segments of this report it will not be expanded upon any further right now. Another risk would be the one of third-party hackers. These hackers could try to get access to the records of users of the measuring device, infringing on their privacy in the process. One question however would be why hackers would be interested in doing this. Of course, gaining information to personal information that can be used for identity theft, fraud or blackmail would be of potential interest to hackers, but there seems to be no practical way in which posture sensor data could be used like this.

Tool 7: closing the loop: ethical feedback and iteration

"Ethics in practice: A toolkit" described ethical design and engineering as a loop that needs to be closed. No successful project aims to remain stagnant and neither should their ethical basis. When a project changes its direction, the ethical considerations that are part of its design should change accordingly. To do this, ethical feedback loops should be designed and implemented into the project. In essence, posture sensors seem to have potential to help spread awareness and knowledge about the quality people's posture and the effects it has on their long-term health. Posture sensors could also be a helpful tool for healthcare professionals to learn more about the human body. It is important, however, that during the development of such tools, one does not lose track of these more altruistic goals in favor of the more capitalistic, profit driven, mentality. As with any health-related wearable, the potential for privacy abuse and false claims about its effectiveness is large. A project like this needs a strong vision of what is important, and what is not. With clear top-down

communication about what direction the project is taking. To achieve this, it is important that one does not lose track of the final goal of the project, namely: helping people improve their posture, and in the long term their wellbeing. To build in an extra safety measure against the loss of a strong, ethically well-founded, vision, it is advised to write down the final goals the technology needs to achieve, and to consult said goals every time a decision needs to be made about the direction the project will take.

[1] S. Vallor, B. Green, and I. Raicu, "Ethics in Tech Practice: A Tooolkit," pp. 1–14, 2018.

[2] "home @ www.fitbit.com." .

[3] "Apple Watch - Close Your Rings - Apple." [Online]. Available: https://www.apple.com/watch/close-your-rings/. [Accessed: 10-Apr-2019].

[4] "strap-on-the-fitbit-john-hancock-to-sell-only-interactive-life-insurance-idUSKCN1LZ1WL @ www.reuters.com.".